全美经典
学习指导系列

SQL 编程
习题与解答

FUNDAMENTALS OF SQL PROGRAMMING

最佳的复习资料，实用的辅助教材

与国外高校计算机水平保持同步

为考研和出国深造奠定坚实基础

Ramez A. Mata-Toledo　Pauline K. Cushman 著
胡志君 高燕林 等译

全球销售超过3000万册！

机械工业出版社 China Machine Press
中信出版社

全美经典学习指导系列是一套快捷有效的学习指南，该套丛书针对各专业的技术重点提供了数百个实例、习题及答案，通过这些实战练习，不但可以洞悉各门技术精髓，而且能够使考试成绩大幅攀升，更会助你与国外大学生的计算机水平看齐，为将来考研或出国深造奠定坚实基础。

全美经典学习指导系列深得高校学生的喜爱。由于有了这套丛书，在历年的专业考试中，成千上万的学生获得了优异成绩。

想成为一名优等生吗？——请选择全美经典学习指导系列！

如果时间不允许却想成绩骄人，这本书可以助你：

- 通过具体范例解决疑难问题
- 考前快速强化
- 迅速找到答案
- 快捷而高效地学习
- 迅速掌握技术重点，无需翻阅冗长的教科书

全美经典学习指导系列以方便快捷的形式提供了考生需要了解的信息，同时不致使你淹没在不必要的细节当中。另外，还可以通过大量的编程练习来测试所学的技巧。该丛书可以与任何教材配合使用。使学生们能够根据各自的进度来学习，从而获得事半功倍的效果！**全美经典学习指导系列**的内容系统而完备，是毕业考试和专业考试的理想参考书。

本书包括：

- 全面概括了最流行数据库的通信语言
- 对最重要的数据库开发者使用的语法进行了简要的解释
- 在SQL程序设计中有200多个解答范例，包括详细步骤的说明
- 例题及习题解答会帮助你掌握SQL程序设计的基础

如果想获得优异成绩并且能够全面掌握**SQL**编程技术，本书是不可或缺的最佳辅导老师

C++ 编程习题与解答

C++ 编程习题与解答（英文版·第2版）

Java 编程习题与解答

Java 编程习题与解答（英文版）

SQL 编程习题与解答

Visual Basic 编程习题与解答

操作系统习题与解答（英文版）

关系数据库习题与解答

关系数据库习题与解答（英文版）

计算机导论习题与解答

计算机导论习题与解答（英文版）

计算机体系结构习题与解答（英文版）

计算机图形学习题与解答

计算机图形学习题与解答（英文版·第2版）

计算机网络习题与解答

软件工程系统习题与解答

数据结构习题与解答

——Java语言描述

数据结构习题与解答

——Java语言描述（英文版）

数据结构习题与解答

——C++语言描述（英文版）

ISBN 7-111-10851-5

9 787111 108511

计算机图书 江西版

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037

购书热线：(010)68995259, 8006100280

(北京地区)

总编信箱：chiefeditor@hzbook.com

华章图书

ISBN 7-111-10851-5/TP 2503

定价：29.00元

全美经典
学习指导系列

SQL编程习题与解答

FUNDAMENTALS OF SQL PROGRAMMING

Ramon A. Mata-Toledo　Pauline K. Cushman 著
胡志君 高燕林 等译

机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

本书主要是根据最新国际标准SQL92，为想学习SQL语言的人们编写的。书中用大量的例题讲解了SQL命令的使用及注意事项，对于从事数据库开发和进行电子商务设计的工程技术人员来说，是一本很好SQL/92参考手册；对于自学者更是一本难得的SQL自学教材，它可以帮助你在考试中取得好成绩。

本书版权登记号：图字：01-2002-0355

图书在版编目（CIP）数据

SQL编程习题与解答/（美）托勒多（Toledo, P. A.）等著；胡志等译. -北京：机械工业出版社，2002.8

（全美经典学习指导系列）

书名原文：Fundamentals of SQL Programming

ISBN 7-111-10851-5

Ⅰ.S… Ⅱ.①托… ②胡… Ⅲ.SQL语言－程序设计－解题 Ⅳ.TP312-44

中国版本图书馆CIP数据核字（2002）第063173号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码100037）

责任编辑：华章

北京忠信诚印刷厂印刷·新华书店北京发行所发行

2002年8月第1版第1次印刷

787mm×1092mm 1/16·18.25印张

印数：0 001-5 000册

定价：29.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

作 者 序

本书是针对想学习 SQL 语言的人们编写的。SQL 语言是用于与关系数据库管理系统进行通信的标准计算机语言，虽然不同的数据库管理系统的开发商使用不同的 SQL 版本，但我们一般还是尽可能使用通用的 SQL。为了便于说明，本书使用在 Windows 95/98/NT 下运行的 Personal Oracle 8i 作为数据库管理系统，这不仅因为该系统已被广泛使用，还因为它易于获得。读者可从 Oracle 公司的网站（www.oracle.com）免费下载该系统。本书所用的全部 SQL 代码均可在装有 Oracle 的平台上工作。此外，为了便于阅读，本书的所有源程序均可从 www.cs.jmu.edu/sqldata/ 下载。

本书可与市场上销售的大部分通用的数据库书籍一同使用，同时它是我们所推出的 Schaum 系列丛书《关系数据库习题与解答》一书的姊妹篇。

感谢 McGraw-Hill 的全体员工的支持和帮助，特别感谢我们的责任编辑 Babara Gilson。

希望本书对 SQL 语言及关系数据库的介绍能为读者提供帮助。享受 SQL 吧！

目　录

第 1 章　SQL 导论及关系数据库的概念

1.1　SQL 语言

SQL 语言是用于和关系数据库管理系统（RDBMS）进行通信的标准计算机语言。SQL 标准由国际标准化组织（International Standards Organization，ISO）及美国国家标准协会（American National Standard Institute，ANSI）定义。该语言的正式名称为国际标准数据库语言 SQL（1992），这个标准的最新版本通常称为 SQL/92 或 SQL2。本书中，我们将该标准称为 ANSI/ISO SQL 标准或简称为 SQL 标准。SQL 语言最初定义于 20 世纪 70 年代早期，那时叫 SEQUEL，是 Structured English QUEry Language 的缩写。SEQUEL 首先作为系统 R 的一部分实施。系统 R 是 IBM 关系数据库管理系统的一个原型。SEQUEL 名称中的单词 English 最终被去掉，这样就缩写为 SQL。Oracle 公司（最初的名称为 Relational Software Inc.）于 1979 年推出了该语言的第一个商业版本。

大多数的关系数据库开发商支持 SQL/92，但并不是百分之百地符合这个标准。目前市场上存在一些不同风格的 SQL，因为每一个 RDBMS 开发商都试图拓展这个标准，以增加产品的吸引力。在本书中，我们尽可能使用 SQL/92 标准。但我们通过 Personal Oracle 8i（Oracle 关系数据库管理系统的 PC 版本）阐述这些特性，对于那些明显拓展或偏离标准之处，我们将指出该语言的一些专有版本的差异或兼容性。

SQL 语言的一个主要特点是：它是一个说明语言，也可以称为非过程语言。对编程者来说，这意味着不用一步一步地详述计算机为获得特定结果而需执行的所有运算，而是由编程者指明数据库管理系统需要完成的任务，然后让系统去自行决定如何获得想要得到的结果。

SQL 语言的语句或命令也称为数据子语言，通常分为两大类。每一个子语言与 SQL 语言的某一方面相关，其中一类是数据定义语言（data definition language[①]，DDL），包括一些支持定义或建立数据库对象（如表、索引、序列及视图）的语句，最常用的 DDL 语句为不同形式的 **CREATE**、**ALTER** 及 **DROP** 命令。在后面的相关章节中，我们将详细讨论每一个语句。另一类子语言为数据操纵语言（data manipulation language，DML），包括允许对数据库对象进行处理或操纵的语句，最常用的 DML 语句为不同形式的 **SELECT**、**INSERT**、**DELETE** 及 **UPDATE** 语句，这些语句也将在后面讨论。需要注意的是，在一个数据库中建

① 数据定义语言在 ANSI 中称为模式定义语言。

立的全部对象都保存在数据字典或目录中。

SQL 语言能够以交互方式使用或将它嵌入表单中。交互式 SQL 允许用户对数据库管理系统直接发出命令[①] 并且收到运行后的结果。当使用嵌入式 SQL 时，SQL 语句包含在用通用的语言（如 C、C++ 或 COBOL 语言）编写的程序中，此时我们将通用的编程语言称为主语言（host language）。使用嵌入式 SQL 的主要原因是为了使用附加的程序设计语言的特性，而这些特性通常不被 SQL 支持

当使用嵌入式 SQL 时，用户不能直接观察到不同的 SQL 语句的输出，结果以变量或过程参数的形式返回。

任何可交互式使用的 SQL 指令，均可用作为应用程序的一部分，这是一条总原则。但用户必须记住，当 SQL 语句交互式使用或嵌入一个程序时，可能存在一些语法的变化。本书中我们仅考虑 SQL 的交互式形式。

因为 SQL 专门与关系数据库管理系统一起使用，所以对这种数据库有一个基本了解是必要的，这样可以更好地理解这种语言的不同特性。接下来我们将讨论关系数据库管理系统。

1.2 关系数据库管理系统

允许用户定义、建立及维护一个数据库，并对数据提供可控式存取的软件系统称为数据库管理系统（database management system，DBMS）。数据库一般指数据本身，但在一个计算机化的数据库中，DBMS 还包括一些附加组件，如硬件、软件本身及用户。用户可根据需要使用数据库管理系统提供的应用及接口来存取或检索数据。用户使用恰当的用户名及口令登录到 RDBMS 后，可与 RDBMS 进行通信。登录成功后，用户可开始对话。当用户注销或与数据库断开连接后，对话结束。附录 A 演示了如何登录到 Personal Oracle Edition 8 数据库。

在数据库中描述现实的集合称为个体域（universe of discourse，UOD），UOD 只包括这样一些现实，它们由逻辑上具有凝聚性并与用户紧密相关的一批事物构成。因此，数据库应该面向特定的用户或为特定目的不断地设计、建立及填充。

基于关系数据模型的数据库管理系统称为关系数据库管理系统，简写为 RDBMS。在这种类型的数据库中，构成 UOD 的所有信息以关系表示。虽然关系可用数学术语来定义，为了使用方便，我们将关系视为二维表。表或关系是拥有数据的数据库对象。在本书中，术语表及关系可互换。每一个关系均由一个关系名以及列或属性的集合组成，表中的数据以行或元组的集合出现。在关系中的属性总数称为关系的度。任何一次出现在关系中的行的总数称为关系的基数。本书中的术语列及属性可互换，同样我们将术语行及元组视为同义。在老的系统中，字段及记录分别与属性及行同义。

① 一些作者将用在交互式形式中的指令称作命令，将嵌入指令称作语句。在本书中这两个术语可互换。

依据 ANSI/ISO SQL 标准要求，在每个关系中每一个属性列必须有一个名称，且在同一表中所有的属性列名必须不同。但在两个不同的表中，两列可以有相同名称。至于关系的度或基数，SQL 标准对一个表中属性列的总数或行的总数并未限制，但制造商往往会对属性列的总数加以限制，而不对行的总数加以限制。从操作上看，一个表不含任何行是可行的，即形成的是一个空表，但表中至少必须有一个属性列。

例 1.1 中的图 1-1 显示的是名称为 CUSTOMER_ORDER 关系的二维表的基本形式。

例 1.1

写出图 1.1 中 CUSTOMER_ORDER 关系的度及基数。

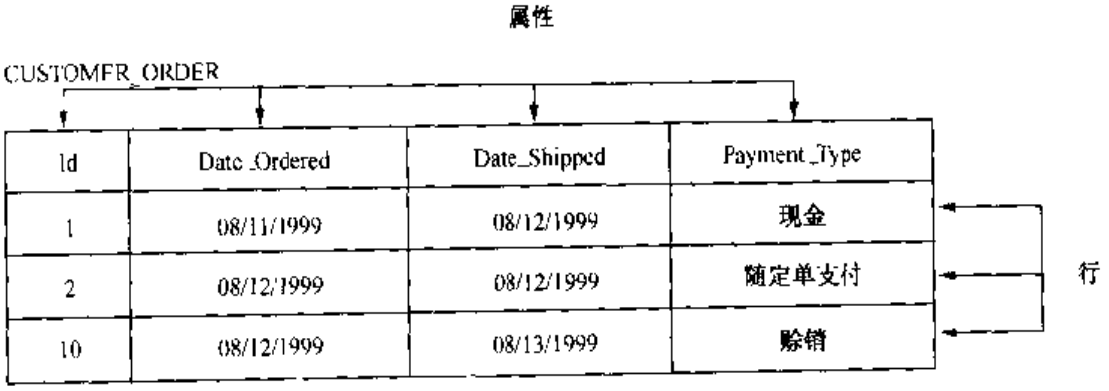


图 1-1　CUSTOMER_ORDER 关系的二维表

图 1-1 表明 CUSTOMER_ORDER 关系由四个属性列组成：Id、Date_Ordered、Date_Shipped 及 Payment_Type，表中有三个不同的行。根据前面讲的术语，称CUSTOMER_ORDER 关系的度为 4，基数为 3。

在本书中，我们假定一个关系的每项至多有一个单一值，即每行及每列相交处最多有一个单一值[①]。对于任何给定的关系 r，r 中的任意属性 A 及 r 中任意元组 t，我们可用符号t(A)表示元组 t 在属性列 A 下的值，该值在列 A 和行 t 的交点。

例 1.2

在图 1-1 的 CUSTOMER_ORDER 关系中，如果 t 是表中的第一行，A 是关系中的任意列，写出各个 t(A)值分别是多少？如果 t 是表中的第一行，则 t(Id) = 1，t(Payment_Type) = 现金，t(Date_Ordered) = 08/11/1999，t(Date_Shipped) = 08/12/1999。

在 CUSTOMER_ORDER 关系中，每一个属性列有一个相关的域，通常表示为 Domain(属性)。域等同于出现在关系 r 中的特定列的值的集合，即对于关系 r 中的任一元组 t 及任一属性 A，t(A)必须是 Domain(A)的一个元素，用数学术语中的集合理论，可表示为 $t(A) \in Domain(A)$。

① 这保证了关系属于第一规范式，即 1NF。

例 1.3

确定图 1-1 的 CUSTOMER _ ORDER 关系表中不同属性的可能域。

在本例中，我们假定属性 Id 的域为正整数集，属性 Date _ Ordered 及属性 Date _ Shipped 的域为有效日期集，属性 Payment _ Type 的域则为包括以下值的有限长度字符串的集：{现金、赊销、邮政汇票、支票、随定单支付、信用卡、借记卡、未知}，由此我们可以如下表示这些域：

Domain (id) = 正整数集

Domain (Payment _ Type) = {现金，赊销，邮政汇票，支票，随定单支付，信用卡，借记卡，未知}

Domain (Date _ Ordered) = Domain (Date _ Shipped) = 有效日期集

注意，在 CUSTOMER _ ORDER 关系中，对每行 t 及属性 A 而言，t (A)是它对应域的一个元素。

1.3 关系的候选码及主码

在关系模型中，码是一个基本概念，在数据库中它可提供检索元组的基本机制。假设有一个关系 r 及其属性 A_1，A_2，A_3，……，A_n，只要 K 满足以下条件，我们可将这些属性的任意子集 K = {A_1，A_2，……，A_k} 称为候选码：

1. 对于关系 r 中任意两个不同的元组 t_1、t_2，存在子集 K 的属性 B，且 $t_1(B) \neq t_2(B)$，这表明 r 的两个不同元组在 K 的所有属性中均无相同的值，这个条件就是码的惟一性特性。
2. K 的子集中不存在满足惟一性特性的子集 K′。换句话说，删除 K 中的任何元素都会破坏惟一性特性，这个条件就是码的最小性特性，该特性保证了组成码的属性个数最少。

在一个关系 r 中，存在多个候选码，故可将其中的任意一个候选码定为关系中的主码(primary key，PK)。主码的值可用作该关系的寻址机制。主码一经选定，其他候选码有时也称为替代码。对每个表而言 RDBMS 只允许有一个主码。主码可由一个单一属性组成(单一主码)，也可由多个属性组成(组合主码)。在本书中，我们将对主码中的属性用下划线标志出来。在例 1.1中，属性 Id 是 CUSTOMER _ ORDER 关系中的主码(注意 Id 已经有下划线)。既然 Id 已是 CUSTOMER _ ORDER 关系中的主码，那么没有两份订单具有相同的 Id 值。

因为主码用于确定一个关系中元组或行的惟一性，故构成主码的属性值不可为空(NULL)。这样对主码就产生了附加的限制，称为完整性约束。一个空值表示无信息、未知数或不恰当的数据。读者必须记住一个空值既不是一个零值，也不代表计算机中的一个特定值[①]。

① 在 SQL 中，空值和其他值之间的大多数比较是通过定义未知而非真或假来进行的。

例 1.4

DEPT 表及一些行如下所示，试解释这些行能否插入 DEPT 表。

DEPARTMENT	NAME	LOCATION	BUDGET
10	Research	New York	1500000
	Accounting	Atlanta	1200000
15	Computing	Miami	1500000

DEPT

DEPARTMENT	NAME	LOCATION	BUDGET
20	Sales	Miami	1700000
10	Marketing	New York	2000000

10	Research	New York	1500000	← 该行不能插入 DEPT 表。它违反了码的惟一性特性，因为表中已存在部门 10
	Accounting	Atlanta	1200000	← 该行不能插入 DEPT 表。它违反了码的完整性约束，因为部门码不能为空 (NULL)。
15	Computing	Miami	1500000	← 该行可插入 DEPT 表。因为它不违反任何约束。

1.4　外码

因为具有相同基础域的列也可用在数据库的关系表中，外码(foreign key，FK)的概念使 DBMS 维持了两个关系的行间或同一关系的行间一致性。该概念可正式定义如下：假设在同一数据库中已知有关系 r_1 及关系 r_2[①]，如果同时满足以下两个条件，则关系 r_1 的 FK 属性组可称为 r_1 的外码：

- FK 中的属性与关系 r_2 中已定义为 r_2 的 PK 属性组具有相同的基础域，则 FK 是以关系 r_2 的 PK 属性为参照。
- 关系 r_1 中任何元组的 FK 值或为 NULL，或为关系 r_2 中一个元组的 PK 值，两者必居其一。

在实际操作中，外码概念保证了：关系 r_1 的元组若引用关系 r_2 的元组，关系 r_2 的元组必须已经存在。对外码的这个约束称为参照完整性约束。一些作者将包含外码的表叫子表，包含参照属性的表叫父表。应用这些术语，可以称一个子表的每一行的 FK 值或为 NULL，或与父表的一个元组的 PK 值(或 UNIQUE 值)相对应。

① 按照外码的这个定义，关系 r_1 和 r_2 可以是同一个关系。

例 1.5

假设下表中属性 EMP _DEPT 是 EMPLOYEE 表的一个外码，其中 EMPLOYEE 表以 DEPARTMENT 表的属性 Id 为参照。请指出下面给出的各行是否可插入 EMPLOYEE 表。

ID	NAME	LOCATION
10	Accounting	New York
40	Sales	Miami

DEPARTMENT

EMP ID	EMP_NAME	EMP_MGR	TITLE	EMP_DEPT
1234	Green		President	40
4567	Gilmore	1234	Senior VP	40
1045	Rose	4567	Director	10
9876	Smith	1045	Accountant	10

EMPLOYEE

9213	Jones	1045	Clerk	30
8997	Grace	1234	Secretary	40
5932	Allen	4567	Clerk	NULL

该行不能插入 EMPLOYEE 表。它违反了码的参照完整性约束。在部门表中没有部门 30。

它没有违反任何约束，该行可以插入 EMPLOYEE 表。

该行可插入 EMPLOYEE 表。在 EMP_DEPT 列中NULL 值是可接受的。NULL 值可能表示该雇员尚未分配到某一部门。

注意：在上例中，我们使用关键字[①] NULL 表明部门的缺乏状态，而在例 1.4 中，DEPARTMENT 的输入保留了空白。读者必须注意到在某些系统中，允许以两种方式表示空值，而在另一些系统中，只允许以关键字 NULL 表示空值。

1.5 关系运算符

关系运算符是一组能够使我们对数据库的表进行操纵的运算符，它们具有封闭特性(closure property)，因为能根据关系操作产生新的关系。当某个关系运算符用于操纵一个关系时，可称该运算符被应用于关系。在本部分仅讨论选择、投影及等值连接关系运算符。在第 2 章，我们将说明在 SQL 中如何使用这些运算符以及一些辅助的关系运算符。

① 关键字指对系统有特殊意义的单词，不能在特定内容之外使用。

1.5.1　选择运算符[①]

当选择运算符用于关系 r 时，这个运算符会产生另一个关系，它的行是满足指定条件的在关系 r 中行的子集，所产生的关系和 r 具有相同的属性。我们可以给这个运算符一个比较正规的定义：假设 r 是一个关系，A 是 r 的一个属性，a 是 Domain (A)的一个元素，在属性 A 上 r 的选择，即产生 r 的 t 元组子集且 t (A) = a。在 A 上 r 的选择可表示为 $\sigma_{A=a}(r)$。下例说明了选择运算符是如何工作的。需要注意的是：选择运算符是一个单目运算符，即一次只对一个关系进行运算。

例 1.6

在上例的 EMPLOYEE 关系中，找出所有在部门 10 工作的雇员的信息。

因为我们需要检索所有在部门 10 工作的雇员信息，所以必须确定 $\sigma_{EMP\_DEPT=10}(EMPLOYEE)$。注意此例中关系 EMPLOYEE 的元组需满足的条件是 t (EMP _ DEPT) = 10。产生的表如下所示：

$\sigma_{EMP\_DEPT=10}(EMPLOYEE)$

EMP_ID	EMP_NAME	EMP_MGR	TITLE	EMP_DEPT
1045	Rose	4567	Director	10
9876	Smith	1045	Accountant	10

1.5.2　投影运算符

投影运算符也是一个单目运算符，不同之处在于：选择运算符选择关系中行的子集，而投影运算符选择列的子集。该运算符的正式定义如下：关系 r 对其属性组 X 的投影可表示为 $\pi_X(r)$。它是一个新的关系，可以通过首先排除关系 r 中未在 X 中指出的列，然后排除任意重复元组来获得。在下面的例子中，我们将对此进行说明。

例 1.7

在下面的 DEPARTMENT 表中，指出不同部门分别位于何地？根据这个回答，我们能否说出在这个表中有哪些不同的地点？

① 选择运算符也可称为 Restrict 或 Select 运算符，我们不使用单词 select 是为了避免与 SQL 命令中的 select 相混淆。

DEPARTMENT

ID	NAME	LOCATION
10	Accounting	New York
30	Computing	New York
50	Marketing	Los Angeles
60	Manufacturing	Miami
90	Sales	Miami

由于我们需要决定目前在 LOCATION 列中出现的不同值的数量，因此需在 DEPARTMENT 表的 LOCATION 属性上做投影，即需要找出 $\pi_{LOCATION}$(DEPARTMENT)，此时，r= DEPARTMENT, X= {LOCATION}，可得到如下关系：

$\pi_{LOCATION}$(DEPARTMENT)

LOCATION
New York
Los Angeles
Miami

注意，在结果表中，New York 及 Miami 只出现了一次。也可观察到 LOCATION 是此表中的惟一属性。

总体说来，关系在属性 LOCATION 上的投影，不能告诉我们 DEPARTMENT 表中目前出现的地点总数，因为重复值已被去掉。在投影关系中只有三个地点，而 DEPARTMENT 表中地点的总数是 5。

例 1.8

使用上例的关系表，指出不同的部门及其所在地。

此例中，X= {NAME, LOCATION}，r=DEPARTMENT，因此可得出以下关系：

$\pi_{NAME,LOCATION}$(DEPARTMENT)

NAME	LOCATION
Accounting	New York
Computing	New York
Marketing	Los Angeles
Manufacturing	Miami
Sales	Miami

由于部门名称及其所在地的组合是惟一的，因此结果关系中不含重复值。注意，(Manufacturing, Miami)及(Sales, Miami)是不同的元组，同样，位于纽约的不同部门也是不同的元组。

1.5.3　等值连接运算符

等值连接运算符[1]是二元运算符，它可将两个关系(不一定相同)组合到一起。通常，这个运算符通过两个关系中的共同属性将它们连接到一起，即来自第一个关系的元组与第二个关系的元组对某一共同属性 X 有等值时，将它们并置可得到全部元组的连接。从数学上可如下定义：设关系 r 含有属性组 R，关系 s 含有属性组 S，并且 R 及 S 有一些共同的属性[2]，设关系 R 和 S 共同的属性组为 X，可表示为 $R \cap S = X$，r 与 s 的连接(可表达为 r Join s)是一种新的关系，其属性是 $R \cup S$ 的元素。此外，对 r Join s 关系中的每一元组 t，需同时满足以下三个条件：(1)关系 r 中的某一元组 t_r，满足 $t(R) = t_r$；(2)关系 s 中的某一元组 t_s，满足 $t(S) = t_s$；(3) $t_r(X) = t_s(X)$。下例对该运算符如何起作用进行了说明。

例 1.9

根据下面所示的两个表的共同属性 DEPT 连接这两个表。写出该操作结果可满足哪些可能的用户请求？这两个表能用属性 ID 连接吗？

DEPARTMENT

ID	DEPT	LOCATION
100	Accounting	Miami
200	Marketing	New York
300	Sales	Miami

DEPARTMENT

ID	NAME	DEPT	TITLE
100	Smith	Sales	Clerk
200	Jone	Marketing	Clerk
300	Martin	Accounting	Clerk
400	Bell	Accounting	Sr. Accountant

在本例中 DEPT 是两表的共同属性，这两表的连接可表示为 DEPARTMENT **Join** EMPLOYEE(如下表)。由于两个表均有属性 ID，为了避免 DEPARTMENT 表中的属性 ID 与 EMPLOYEE 表中的属性 ID 相混淆，在连接两表时，需要将此属性与它相应的表名一起列出。注意：这一点与前面所讲的“表的列名称不能相同”的相一致。还有一点需注意，两个表共同的列连接后，不再重复给出。该连接操作的结果可以满足用户“显示有关雇员的所有信息及他们所属部门 ID、部门名称及部门所在地”的请求：

---

[1] 此类型的连接也称为自然连接

[2] 共同属性并不要求两表中有相同名称的属性，但它们的基本含义和域必须相同。

DEPARTMENT Join EMPLOYEE

DEPARTMENT ID	DEPT NAME	LOCATION	EMPLOYEE ID	EMPLOYEE NAME	TITLE
100	Accounting	Miami	300	Martin	Clerk
100	Accounting	Miami	400	Bell	Sr. Accountant
200	Marketing	New York	200	jones	Clerk
300	Sales	Miami	100	Smith	Clerk

DEPARTMENT 表及 EMPLOYEE 表不能由属性 ID 连接，因为 DEPARTMENT 表的属性 ID 与 EMPLOYEE 表的属性 ID 具有不同含义，一个是部门的 ID，一个是雇员的 ID。这表明两个表不能仅根据它们的属性具有相同的名称而将两个表连接起来。两个表必须根据它们的共同属性连接，这些共同属性必须有相同的域及相同的意义。

在第 2 章中讨论关系运算符在 SQL 中的执行情况时，将对此进一步讨论。

1.6 属性域及其执行

如前所述，属性域定义了一个表中包含的列值的特性。在任一 RDBMS 中，任一给定的属性域用一个数据类型来实现。标准 SQL 语言命名并定义了一系列的基本数据类型。虽然大多数的 RDBMS 开发商均支持这些数据类型，但它们在各个开发商的实现细节上并不相同。因此建议用户查询相关的 SQL 参考手册，以确定 RDBMS 开发商实现的数据类型特点。表 1-1 列出了一些基本的 SQL 数据类型及几个 RDBMS 开发商的实现。

表 1-1 某些标准 SQL 数据类型及某些 RDBMS 开发商的实现

标准 SQL	Oracle	Access	DB2
Character (n)：n 是指字符数	Char (n)：固定长度的字符，最大值为 255 个字符	Text：固定长度的字符，最大值为 255 个字符	Character (n)：与 Oracle 的 Char (n)相同
Character varying (n)：n 为可存储于列中的最大字符数	Varchar2 (n)：可变长度的字符。最大值为 2000 个字符	Text：可变长度的字符，最大值为 255 或 Memo 可变长度的字符，最大值为64 000	Varchar (n)：与 Oracle 的 Varchar2 (n)相同
Float (p)，p 是数字的总个数	Number：数的大小在 1.0×10^{-130}和 38 个 9 后接 88 个 0	Single 或 Double：取决于数据值的范围	Float：与 Oracle 的 number 相同
Decimal (p,s)：至少有 p 位数字，带有 s 位小数位	Number (p,s)：p 的范围在 1 至 38，而 s 的范围在 -84 至 127 之间	Integer 或 Long Integer：取决于数据值的范围	Integer：与 Oracle 的 number (38)相同

数据类型 character (n)或 character varying (n)的列(此处的 n 指可存储在列中的最大字符数)通常用于含文本的数据或不参与计算的数据。这种数据类型的例子有：名称、地址、社会保障号码及电话号码等。character (n)与 character varying (n)数据类型之间的主要不同之处在于它们如何存储短于列最大长度的字符串(字符的顺序)。当一个字符串少于 n 个字符

时，若存储在 character (n)列中，RDBMS 以空格填充至该字符串之后，以产生一个正好为 n 个字符的字符串。若存储在 character varying (n)列中，RDBMS 按原样存储字符串，不加空格。因此，当我们已知一个字符列的内容长度有变化时，最好将列定义为 character varying 类型，以使这个信息可被 RDBMS 更有效地存储。不管是作为固定还是可变类型存储，文本数据均区分大小写。例如字符串“abc”与字符串“aBc”不同，同样，字符串“ xyz”与字符串“xyz”也不同，因为前者的第一个字符以空格开始。

Float (n)类型的列(此处 n 是数字总数)通常用于表示大的数量级或科学计算的结果。例如我们可使用浮点数据类型来表示 1.0×10^{-10}至 $1.0\times10^{+10}$范围内的数。

Decimal (p,s)类型的列用于表示定点数，精度 p 用于表示十进制小数点左边和右边数字的总位数，s 用于表示十进制小数点右边数字位数。当某数是一个整数时，s 为 0。例如，用这种数据类型表示数 123.23，就是 Decimal (5,2)；用这种数据类型表示数 125，就是 Decimal (3,0)。

表 1-1 未给出的 RDBMS 的另一种常用数据类型是 DATE。该数据类型允许用户存储日期和时间信息。日期常常以缺省格式 DD-MM-YY 出现，其中 DD 代表天，MM 代表月，YY 代表年。在本书的第 6 章里将对一些用于操纵日期/时间信息的内部函数进行说明。

1.7　数据库对象的命名约定

数据库对象的表名及属性名必须服从某些规则或约定，在不同的 RDBMS 中，这些规则或约定可能是不同的。如果不遵守特定的 RDBMS 命名规则就会出错。但用户如果能遵守下列原则，一般来说是安全的：

- 名称在 1 到 30 个(在 MS Access 中是 64 个)字符长度范围内。当然，也有例外情况，如 Oracle 数据库要求数据库的名称必须限制在 8 个字符以内。
- 名称必须以字母开头(小写或大写均可)，其余字符可为大写、小写字母、数字或下划线字符的任意组合。

1.8　SQL 语句的结构及 SQL 书写指导

每个 SQL 语句或命令均为一个或多个子句的组合。子句通常由关键字引入。图 1-2 给出了 SQL 语句的一个例子。读者目前不必关心这个语句内部是如何运行的，只需注意它的结构特点即可。

```
SELECT column-name-1, column-name-2, . . . column-name-N
FROM table-name
WHERE Boolean-condition
ORDER BY column-name [ASC|DESC] [, column-name [ASC|DESC], . . .];
```

图 1-2　SQL 语句结构中的关键字及子句

在这个SQL语句中，有四个关键字及四个子句。关键字在图1-2中以粗体字显示。前面已讲过，关键字在语言中是有特定意义的单词。只使用关键字而无特定的上下文会产生错误。在图1-2中四个子句都有下划线。注意每个子句以关键字开始[1]。

在这个语句中，前两个子句(SELECT及FROM)是必需的，而后两个子句(WHERE及ORDER BY)是可选的。当描述SQL语句的语法时，我们将可选的关键字或子句放在方括号里。根据这个规则，上述语句可重新写成如图1-3所示的样子。

```
SELECT column-name-1, column-name-2, ... column-name-N
FROM table-name
[WHERE condition]
[ORDER BY column-name [ASC|DESC] [, column-name [ASC|DESC] ...]];
```

图1-3 SQL语句中必需及可选子句的明确表示

注意，在ORDER BY子句中，我们将单词ASC及DESC置于一个方括号里，且当中以“|”字符隔开，这个字符有时叫管道。用“|”隔开可供选择的不同选项。用户在写SQL语句时，在每一组选项中只能选择一项。在此图中还需注意带有下划线的单词ASC，它表明这个单词是一个缺省值——即当用户在一系列选项中不进行选择时，系统将自动使用的一个值。在图1-2中，不管何时，当使用ORDER BY子句但用户未选择DESC选项时，RDBMS将缺省使用ASC选项。

在书写SQL语句或命令时，遵循某些规则及指导可以提高语句的可读性并有助于对它们进行编辑。以下是一些指导，读者应牢记于心：

- SQL语句不区分大小写，但子句的第一个关键字通常是大写，以提高SQL语句的可读性。
- SQL语句可分一行或多行书写，但习惯上将一个子句放在它所在的一行中书写。
- 关键字不能跨行分开书写，也不可缩写，极少数情况除外。
- SQL语句以分号(;)结束。分号必须接在该语句的最后一个子句后，但不必在同一行上。

1.9 通过SQL*Plus与Oracle RDBMS交互

在本书中，我们将着重讲述交互式SQL。因此用户必须先进入一个指定的RDBMS，并可在其中执行SQL语句。本书以Personal Oracle版本8i(P08)作为主要的RDBMS。Personal Oracle是Oracle RDBMS的PC版本，选择P08是因为Oracle在工业界及学术界均被广泛使用，而且可直接从Oracle公司的网站上(www.oracle.com)免费下载。此时你应该已经熟悉附录A的内容，其中讲解了登录P08的操作过程。如果情况并非如此，那就要记住，除了附录E外，本书中的概念可普遍应用于大多数的SQL版本。

① 附录B示出一部分SQL关键字的列表。

SQL * Plus 是允许用户与 POS 交互操作的工具，它是识别并执行 SQL 语句的 Oracle RDBMS（服务器）工具。SQL * Plus 既不是 SQL 的延伸，也不是 SQL 的超集，然而，它包括附加的命令，以控制环境设置，访问远程数据库和完成其他一些通用的功能，如有利于格式化查询结果及报告的功能。图 1-4 显示了 SQL * Plus 是如何与 POS 进行交互操作的。

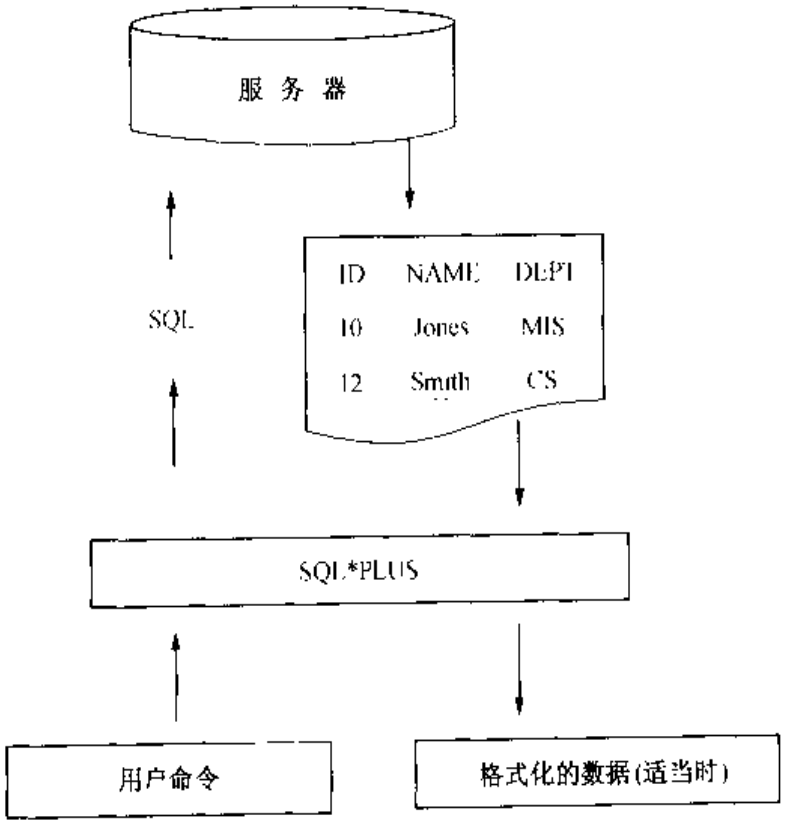


图 1-4　使用 SQL * Plus 与 Oracle 服务器交互

从语法上来看，SQL 和 SQL * Plus 命令并无差别，但在实际操作中，这两类命令之间的差异使我们能对它们进行区分。用户键入的全部 SQL 命令都进入一个称为 SQL 缓冲器的内存区。而 SQL * Plus 命令不存储在 SQL 缓冲器内。在 SQL * Plus 命令中，List 命令（或缩写为 l）使我们能查看临时缓冲器的内容。临时缓冲器一次只能存储一条 SQL 命令，直到新的命令输入为止。SQL * Plus 命令不以分号结束，这与 SQL 命令不同。当敲完一个命令后，用户需要按 Enter 键即可执行它。由于用户往往容易忘记在命令的最后一个子句后敲一个分号，因此，用户可以通过键入“/”，然后按 Enter 键执行存储在缓冲器中的命令。

除非特别强调，否则本书的全部数据库对象都将通过 SQL * Plus 及 POS 创建。

1.10　创建表

在任何 RDBMS 中，表是存储数据的基本单位。它拥有用户可存取的所有数据。要建立一个表，必须有表的名称及组成表的所有属性。此外对每一个属性，用户需定义它的数据类

型。如果需要的话，还可加上一定的约束。表名称使表在 RDBMS 中作为惟一的对象而被标识[①]。列或属性名称用于将一个属性与另一个属性区分开来。属性名在表内必须是惟一的，每个属性的数据类型限定了它的基础域特征。存储在列内的所有值都必须满足列定义的约束条件。

在 SQL 中，用 **CREATE TABLE** 语句或命令即可创建表。图 1-5 显示的是这个命令的基本格式。CREATE TABLE 命令的其他特性将在第 2 章予以解释。附录 C 显示了在本书中使用的 SQL 语言的子集的语法图。我们建议读者通过查询这个附录来学习 SQL 语言的辅助元素。在这节中，我们假定每次创建一个表时，以前同一用户在其模式[②] 中，未以相同的表名创建其他的表。在数据库行话中，由某一用户建立的表为该用户所拥有。

```
CREATE TABLE table-name
(
  column-name-1          data-type-1     [constraint],
  column-name-2          data-type-2     [constraint],
        .
  column-name-N          data-type-N     [constraint]
);
```

图 1-5 CREATE TABLE 命令的基本语法

当描述 CREATE TABLE 命令语法时，称为列定义行的每行格式为：

列名称 数据类型 [约束],

其中可选元素置于方括号内，据此，每一列定义行均需要一个列名及一个数据类型。约束是可选的。通常在敲 CREATE TABLE 命令时，每个列定义行应写在单独的一行，以增加可读性。列定义行以逗号隔开，最后一行后接一个圆括号。对于任意一个 SQL 命令，在圆括号后还应加一个分号。

SQL 标准要求无论何时定义一个约束，必须给约束命名。在建立或修改表时，用户可明确给出约束的名称。否则约束名将由 RDBMS 内部命名。由用户命名的约束称为命名约束；RDBMS 命名的约束由开发商决定，称为未命名约束。虽然约束名可按 1.7 节的习惯命名，但我们对约束命名将使用以下格式：

CONSTRAINT table-name-column-name-suffix

其中 CONSTRAINT 子句是必不可少的，后缀表明约束类型，为一到两个字母序列。表 1-2 列出了将在本书中使用的后缀。未命名约束不能以 CONSTRAINT 子句开头。

① 在用户的表空间、模式或整个系统中，根据该表是否被定义为公用表，通常将表定义为惟一的对象。

② 这个术语指的是数据或模式对象的逻辑结构。每个用户拥有一个模式，其名称属于拥有者。任何有相应权限的用户，可建立自己的模式对象。模式对象有表、同义词、索引、序列及视图。

表 1-2　命名约束的后缀约定

后缀	意义
PK	Primary key（主码）
FK	Foreign key（外码）
NN	Not NULL（非空）
U	Unique（惟一值）

在表 1-3 中给出了本章将考虑的一些约束，若约束为列定义的一部分称为列约束(column constraint)。这种约束类型可在它定义的列上加一个简单的条件。若约束为表定义的一部分时，称为表约束(table constraint)。这种约束类型可用于表中任何列定义一个以上的约束。在第 2 章中我们将讨论表约束。

表 1-3　基本的列和表约束

约束	定义
NOT NULL（*）	防止 NULL 值进入该列
UNIQUE（**）	防止重复值进入该列
PRIMARY KEY（***）	要求进入该列的所有值是惟一的，且不为 NULL

* 列_约束。

** 列_约束或表_约束，分别取决于它是否应用于一个或多个列。

*** 当定义单一属性主码时，是列_约束；当定义组合主码时，是表_约束。

例 1.10

用以下属性及设定建立表 Calling_Card。选择最合适的数据类型。属性：Company_Name、Card_Number、Starting_Value、Value_Left 及 Pin_Number。设定：属性 Company_Name 可具有多达 25 个字符，属性 Value_Left 及 Original_Value 用美元及美分来度量，属性 Card_Number 可具有多达 15 个数字。属性 Pin_Number 限定为 12 个字符。用如下的 SQL 指令建立 Calling_Card 表。

```
CREATE TABLE Calling_Card
( Company_Name          VARCHAR2(25),
  Card_Number           VARCHAR2(15),
  Starting_Value        NUMBER(4,2),
  Value_Left            NUMBER(4,2),
  Pin_Number            CHAR(12)
);
```

属性 Company_Name 显然属于字符类型，并非所有公司名称的字符长度都为 25，故该列的数据类型是 VARCHAR2(25)。Card_Number 列及 Pin_Number 列也都是字符类型，因

为它们不参与任何类型的计算。由于 Card _ Number 的字符长度是可变化的，因此其数据类型为 VARCHAR2 (15)。Pin _ Number 的数据类型为 CHAR (12)，因为该列为定长。Starting _ Value 列及 Value _ Left 列均为数值，可达四位数字，带两位小数。注意为了提高属性名称的可读性，我们使用了下划线字符。

例 1.11

使用未命名约束重新书写上例的 CREATE TABLE，将 Card _ Number 属性定义为主码；将 Pin _ Number 属性定义为惟一码

在本例中，我们可使用一个列约束将属性 Card _ Number 定义为 PK，通过 PK 的定义，该属性也为 UNIQUE，因此没必要以 UNIQUE 来定义它。属性 Pin _ Number 只定义为 UNIQUE 属性。读者需注意这两个约束有一点不同之处：PRIMARY KEY 约束除了要求 Card _ Number 列的值具惟一性外，还要求其值不为 NULL。Pin _ Number 列的 UNIQUE 约束不允许该列中不允许出现重复值，但它允许属性的 NULL 值出现。新的 CREATE TABLE 命令如下：

```
CREATE TABLE Calling_Card
( Company_Name              VARCHAR2(25),
  Card_Number               VARCHAR2(15) PRIMARY KEY,
  Starting_Value            NUMBER(4,2),
  Value_Left                NUMBER(4,2),
  Pin_Number                CHAR(12) UNIQUE
);
```

例 1.12

用命名约束重新书写上例的 CREATE TABLE。

按照命名约束的规定，使用表 1-2 的后缀，与属性 Card _ Number 及属性 Pin _ Number 相关的约束分别为 calling _ card _ card _ number _ PK 及 calling _ card _ pin _ number _ U。

相应的 CREATE TABLE 命令如下：

```
CREATE TABLE Calling_Card
(Company_Name       VARCHAR2(25),
 Card_Number        VARCHAR2(15) CONSTRAINT
                    calling_card_card_number_PK
                    PRIMARY KEY,
 Starting_Value     NUMBER(4,2),
 Value_Left         NUMBER(4,2),
 Pin_Number         CHAR(12) CONSTRAINT
                    calling_card_pin_number_U
                    UNIQUE);
```

注意，书写属性 Card _ Number 及属性 Pin _ Number 的列定义时用了多行，是因为该页的宽度所限。

1.11　描述表的结构

当一个表建立后，可能需要确定表的名称、数据类型及组成表的属性的一些约束。SQL * Plus 命令允许我们通过 DESCRIBE 命令找到这个信息。这个命令的语法如下：

```
DESCRIBE table-name
```

例 1.13

描述表 Calling _ Card 的结构。

满足这个请求的语句如下：

```
DESCRIBE Calling _ Card
```

这个命令的输出结果如下：

```
Name                          Null?          Type
----------------------------  -------------  ------------------
COMPANY_NAME                                 VARCHAR2(25)
CARD_NUMBER                   NOT NULL       VARCHAR2(15)
STARTING_VALUE                               NUMBER(4,2)
VALUE_LEFT                                   NUMBER(4,2)
PIN_NUMBER                    UNIQUE         CHAR(12)
```

DESCRIBE Calling _ Card 命令的输出结果由三列组成，分别为 Name、Null? 及 Type。Name 列由表中不同属性的名称组成。Null? 列指出该属性能否接受 NULL 值。Type 列描述表中属性的数据类型。

1.12　填充表

一个表建立后，用户可使用 **INSERT INTO** 命令将行加至表中。第一次将行加至表中的过程被称为填充。这个命令最简单的形式是允许用户一次向表中添加一行　图 1-6 给出了这个命令的基本语法。在“问题与答案 1.27”中，讲解了 INSERT INTO 命令这种基本形式的另一种变化。在本书的后面部分，我们将说明这个命令的另一种形式，该形式允许我们将一个表的多行插入到另一个表中(见第 7 章)。

在图 1-6 中，column-1、column-2 …… column-N 是表的列，value-1、value-2、value-3 ……value-N 是将被插入到它们对应的列的值。注意，当表建立后，将被插入列的值必须与该列具有相同的数据类型。必须记住的是，对出现在列表中的每一列，在 VALUES 子句中，必须指定一个值。

```
INSERT INTO table-name (column-1, column-2,...column-N)
VALUES (value-1, value-2,...value-N);
```

图 1-6 INSERT 语句的基本格式

例 1.14

将下列数据插入 Calling_Card 表。

ACME	1237096435	20.00	12.45	987234569871
Phone Card, Inc	5497443544	15.00	11.37	433809835833

由于 INSERT INTO 命令的基本形式只允许一次插入一行，故必须使用两个连续的 INSERT INTO 命令，将这两行加到 Calling_Card 表。注意，所有的字符数据都放在了单引号内。

```
INSERT INTO Calling_Card (Company_Name, Card_Number,
                          Starting_Value,Value_Left,Pin_Number)
VALUES ('ACME', '1237096435', 20.00, 12.45, '987234569871');

INSERT INTO Calling_Card (Company_Name, Card_Number,
                          Starting_Value,Value_Left,Pin_Number)
VALUES ('Phone Card, Inc', '5497443544', 15.00, 11.37,
'433809835833');
COMMIT;
```

在最后一个 INSERT INTO 命令后，需加 COMMIT 命令，以确保对表的改动成为永久性的。在下一节我们将讲解 COMMIT 命令。

读者需注意：在 INTO 子句中，列的顺序并不重要，只要与它们相应的值在 VALUES 子句中以同样的顺序出现即可。这样我们可以在一行中以任意顺序来填充列。例 1.15 表明了这一点。

例 1.15

在 Calling_Card 表中插入以下行，并以如下顺序填充列：Pin_Number、Card_Number、Company_Name、Starting_Value 及 Value_Left。

Company_Name	Card_Number	Starting_Value	Value_Left	Pin_Number
ACM	2137096435	20.00	20.00	125234569871
Mobile Phone, Inc	3817443544	20.00	20.00	632809835833

如上例，我们需用两个连续的 INSERT INTO 语句将这两行加至 Calling_Card 表中。新的 INSERT 语句如下：

```
INSERT INTO Calling_Card (Pin_Number,Card_Number,Company_Name,
                          Starting_Value,Value_Left)
VALUES ('125234569871', '2137096435', 'ACM', 20.00,
        20.00);

INSERT INTO Calling_Card (Pin_Number, Card_Number,Company_Name,
                          Starting_Value,Value_Left)
VALUES ('632809835833', '3817443544', 'Mobile Phone, Inc',
       20.00, 20.00);
COMMIT;
```

1.13　COMMIT 命令及 ROLLBACK 命令

对一个表进行 INSERT、UPDATE 或 DELETE 操作后，为了使所做的变更成为永久性的，用户需要提交这些变更。COMMIT 语句允许用户将这些变更永久记录到数据库中。该语句的基本语法是：

COMMIT [**WORK**];

注意，关键字 WORK 是可选的。

读者必须注意：在执行 COMMIT 命令之前，对表中行的所有改变均存储在主存储器的数据库缓冲区或工作区中。如果由于某些原因，用户在提交这些变化前退出了数据库，所以数据库文件中并未写入数据，这些改变也就丢失了。如果对表作改变的用户在一个多用户环境下工作且该表与其他用户共享，则只有该用户对所作的改变发出 COMMIT 命令后，其他用户才可以访问改变后的表。之所以如此是因为用户修改表的行数后，在该用户提交这些变更之前，只有他可对这些行进行排它性存取。所谓“排它性存取”是指其他用户不能察看到目前已改变的内容，相关行被锁定。此时，存取同一表的其他用户并不能察觉到此表已经改变。但当用户提交改变后，修改后的行被写进数据库文件且这些修改行上的锁被打开。在数据被提交后，用户的交互事务开始了，用户可察看到加入了新内容的行。

假设对表的改变（插入、更新或删除）尚未提交，用户可通过 **ROLLBACK** 语句或结束会话的方式，取消对表所作的全部中间变化。上述活动将使 RDBMS 忽略自从用户上一次提交或自从用户开始他或她的交互式会话以来，对任何表、任何数据库对象所作的全部变化。

从技术上来说，**ROLLBACK** 语句是用于取消或终止目前的事务。一个事务是一个逻辑工作单位，通常包括一系列数据库操作。一个事务的所有操作均以组来计算成功或失败，换句话说，就是如果一个操作失败，则所有操作均失败。从这种意义上说，事务是不可分的。此外，事务也是持久的。一旦一个事务被提交，即使出现系统故障，也能保证对元组的改变写入数据库文件。在执行 COMMIT、ROLLBACK 或连接到数据库后，用第一个可执行的 SQL 语句开始一个事务。在执行 COMMIT、ROLLBACK 或从数据库断开后，一个事务结

束。大部分数据库处理完一个 DDL 语句后，发出一个隐式的 COMMIT 语句。

在进行交互会话时，有时希望及时回到某一个特定点(称为保留点)。例如某个用户对一个表作了一些未提交的改变，并意识到最后几个改变不正确或没有必要。如果用户此时发布一个 ROLLBACK 语句，将忽略对表所作的全部改变，包括那些正确的改变。因此对用户来说，如果他或她能回到开始产生不正确改变的数据库缓冲区之前的状态，是再好不过了。能够执行此功能的 SQL 命令是 SAVEPOINT 命令。

只要事务尚未提交，保留点就可使我们返回到事务中的这一点。从这种意义上说，SAVEPOINT 语句在数据库缓冲区中起到书签的作用。保留点能够使用户及时回到特定点，因此能取消日前事务的一部分。

该语句的基本形式如下：

```
SAVEPOINT savepoint-name ;
```

此处，savepoint-name 在事务中是一个惟一的名称。保留点名称通常是单一字母，也可以更长。保留点遵循与数据库中其他对象相同的命名规则。保留点在交互程序设计中很有用，它们能够使用户对程序的执行进行一定程度的控制。若想回到数据库缓冲区的未执行状态之前，可发布以下命令：

```
ROLLBACK savepoint-name;
```

或

```
ROLLBACK TO SAVEPOINT savepoint-name;
```

图 1-7 给出了 SAVEPOINT 命令及 ROLLBACK 命令一起使用的说明。在图 1-7 中的步骤表明了回到保留点可产生什么效果。此时假设用户通过发布一个 SAVEPOINT A 命令开始操作过程，然后插入一个新的元组，并发布 SAVEPOINT B 命令，接着删除表的第一个元组，并发布 SAVEPOINT C 命令。最后用户发布一个 ROLLBACK TO SAVEPOINT A 命令。ROLLBACK TO SAVEPOINT A 命令使用户回到发布 SAVEPOINT A 命令以前的状态。通过返回到保留点 A，用户忽略了在发布 SAVEPOINT A 命令之后对表的所有改变。

显然，本例假设用户在保留点 A 及保留点 C 之间从未发布过一个 COMMIT 命令。如果用户在保留点 C 之后提交数据，则用户不可能回到前面的保留点 A 或保留点 B。同时必须记住的一点是：在回到保留点 A 后，用户不能再改变主意，即不能再重新前进至保留点 B 或保留点 C。

还需要注意的是，保留点名称在一个给定的事务中必须是惟一的。如果用户建立了一个与早先的保留点具有相同名称的保留点，那么早先的保留点就被去掉了。在这些改变提交后，用户可以复用任一早先的保留点的名称，只要该名称在事务中是惟一的即可。

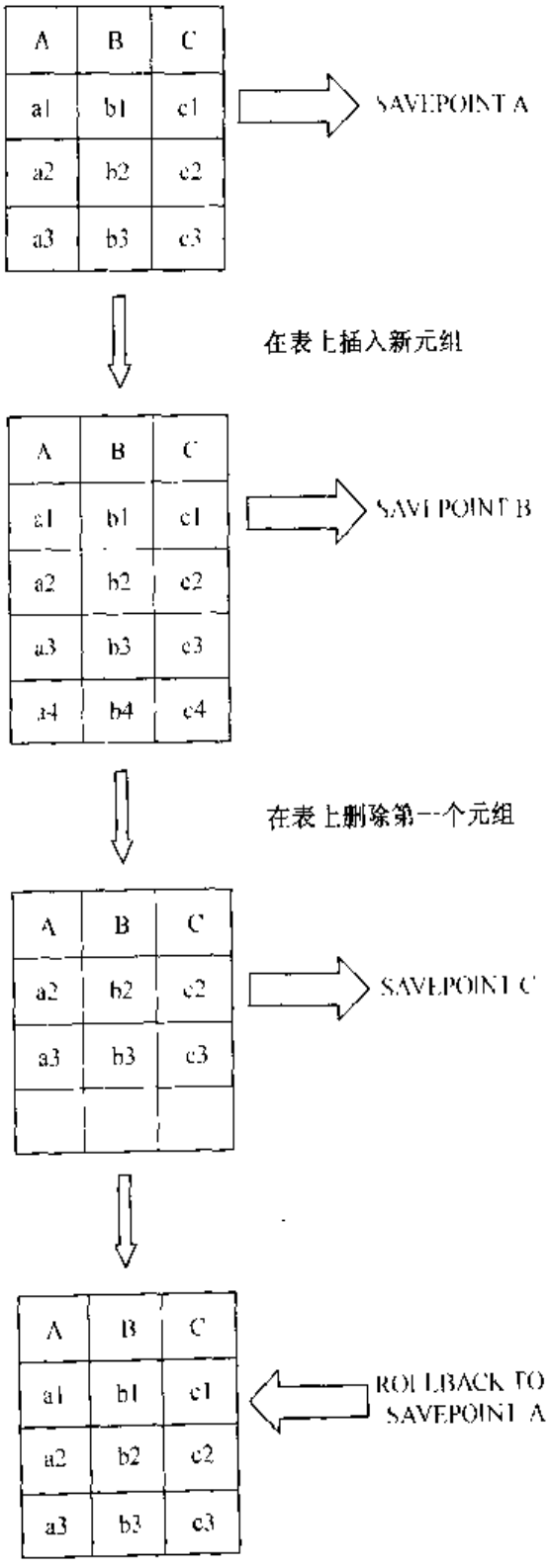


图 1-7　SAVEPOINT 及 ROLLBACK TO SAVEPOINT 语句

1.14　SELECT 语句

SELECT 语句是使用频率最高的 SQL 语句。SELECT 语句主要用于从数据库中查询或检索数据。因此人们习惯于称一个 SELECT 语句为一次查询。在本书中，我们用查询代替

任一 SELECT 语句。这个语句的基本语法如图 1-8 所示。将在后面的章节讨论 SELECT 语句的其他特性。

```
SELECT column-1, column-2, column-3,...,column-N
FROM table-1,..., table-N
[WHERE condition]
[ORDER BY column-1 [ASC|DESC][,column-2 [ASC|DESC]...]];
```

图 1-8 SELECT 语句的基本结构

SELECT 语句至少由两个子句组成：**SELECT** 子句和 **FROM** 子句。**WHERE** 子句和 **ORDER BY** 子句是可选的。SELECT 语句与其他 SQL 语句一样以分号结束。可将上述每一个子句的功能概括如下：

- **SELECT** 子句列出了要显示的列。这个子句中列出的属性是结果关系的列。
- **FROM** 子句列出了应从哪些表获取数据。在 SELECT 子句中提到的列必须是 FROM 子句中列出的表的列。
- **WHERE** 子句说明了条件，即 FROM 子句中给出的表的行需满足的条件。
- **ORDER BY** 子句表示对满足 WHERE 子句的行进行排序的原则或标准。ORDER BY 子句仅对检索的数据的显示有影响，并不改变表中行的内部顺序。

为了帮助用户记住 SELECT 语句的基本结构，一些作者将其功能总结为："从(FROM)表中选择(SELECT)列，其中(WHERE)的行满足一定的条件，对所得到的结果按指定的列排序(ORDER BY)。"

WHERE 子句能够使 SELECT 命令显示满足某个指定条件的数据。例如，你可能想打印出在某一邮政编码区的所有人名，或想看看在某一部门工作的所有雇员的姓名。伴随 WHERE 子句的条件定义了要满足的标准。在本章中，我们考虑的条件类型有以下形式：

```
Column-name Comparison-operator single-value
```

Column-name 是在 FROM 子句中指出的表中的某一列名。Comparison-operator 是在表 1-4中给出的一个运算符。single-value 指一个数值或一个字符串。在本书第 3 章，我们将讨论用布尔运算符 AND、OR 及 NOT 组成的复合条件进行更复杂的查询。

表 1-4 WHERE 子句的比较运算符

比较运算符	说明
=	等于
< >	不等于
<	小于
< =	小于或等于
>	大于
> =	大于或等于

下面例子讲解 SELECT 命令的使用。

例 1.16

使用例 1.9 中的 EMPLOYEE 表，显示在会计部门工作的所有雇员的姓名及职务。

为了从 EMPLOYEE 表检索相关数据，我们使用了以下语句：

```
SELECT name, title, dept ←———— 需要显示的列
FROM employee ←———————————— 从此表中检索数据
WHERE dept = 'Accounting'; ←———— 需满足的标准
```

所得的结果表如下所示：

NAME	TITLE	DEPT
Martin	Clerk	Accounting
Bell	Sr. Accountant	Accounting

注意，在所得结果表中属性名及它们相应值的显示顺序与 SELECT 语句中的顺序是一致的。所有检索到的元组均满足 WHERE 子句的条件。即对结果关系中的任意元组 t，存在 t(DEPT) = Accounting。此外可观察到被检索的行不是按字母顺序排列的。如果我们想让此表按字母顺序排列，可按例 1.17 所示处理。要注意在 SELECT 子句中，属性名是不区分大小写的，即我们将属性大写或小写均可得到相同的结果，但出现在 WHERE 子句中的条件则需考虑大小写，因为 DEPT 是一个字符型列，该条件已被包括在一个单引号中。此外需记住，字符型数据是区分大小写的，如将 WHERE 子句的条件写成 DEPT = 'ACCOUNTING'，则没有元组满足它的条件。注意字符串 'Accounting' 和 'ACCOUNTING' 是不同的字符串。

例 1.17

按雇员姓名的字母顺序显示上一个查询结果。

此时 SELECT 语句需要指示 RDBMS 将结果按字母顺序排序。由于我们欲将结果表按属性 NAME 进行排序，故必须在 ORDER BY 子句中提到此属性。缺省时，行的排序以指定列(在本例中为属性 NAME)的升序排列[①]。能满足此要求的 SELECT 语句如下所示：

```
SELECT name, title, dept
FROM employee
WHERE dept = 'Accounting'
ORDER BY name;
```

结果表即为：

① 升序即按 ASCII 字符的编码顺序从低到高排列。

NAME	TITLE	DEPT
Bell	Sr. Accountant	Accounting
Martin	Clerk	Accounting

注：该表的行按姓氏的字母顺序升序排列

1.15 样本数据库

在本章的余下部分(除非特别说明)，我们将用到运动用品(SG)数据库。附录 D 给出了在 Oracle 中建立本数据库两个不同版本的源程序。本数据库是以 Oracle 公司提供且经常在培训课程中使用的运动用品(SG)数据库为基础而制作的 SG 数据库的修改版本，它得到了 Oracle 公司的许可。

Sporting Goods 是一家美国批发公司，接收来自世界范围内的运动产品零售商店的定单。该公司的客户遍布国内外，每一个客户都有一个惟一的标识号。此外，公司还必须保存有客户的商店名称和电话号码。公司也可保存关于客户的其他信息，如地址、所在城市、州、国家、邮政编码、信誉等级以及对客户喜好的总体评论。一般情况下，客户通过电话或传真订货。公司需跟踪每一份定单的标识号、订货日期及付款方式。订货运到客户手中的日期需保存在数据库中。为了加快所有输入定单的进程，可将客户归入世界上的某一区域，目前有六个区域，分别为中美/加勒比、北美、南美、非洲/中东、亚洲及欧洲，每个区域有惟一的名称及标识号。每个区域有一个仓库，产品由此运至客户手中。SG 具有每个仓库的标识号及其他一些信息，包括地址、城市、州、国家、邮政编码、经理 ID 及电话号码。另外也应该知道库存中每一件商品的惟一的标识号。此外，SG 还必须跟踪产品价格、数量、订购的数量及已发运的数量。为了提高客户满意度，SG 还提供了一条特制产品生产线，对每一种产品，SG 也必须知道它惟一的产品标识号及名称。有时还需要对一个产品及其建议价格、单元销售进行简短描述。

该公司有一些雇员或销售代表了解客户的需求。雇员被派往多个地区，对每一个雇员，公司备有其姓、名、惟一的标识号及计算机用户 ID 信息。其他信息包括该雇员进入公司的时间、评语、职务、工资及佣金百分率。对每一个仓库及它存储的产品，SG 货单记录有每个产品的存货量、再订货点，任何时间的最大存货量，重新进货的日期及必要时所作的无货的解释。表 1-5 至表 1-13 给出了组成 SG 数据库的表的属性及对这些属性的约束。

1.15.1 运动产品数据库表

表 1-5 S_customer 表的属性

列名	描述/数据类型
Id	客户惟一的标识号，最长为 3 个字符
Name	客户名，最长为 20 个字符
Phone	客户的电话号码，最长为 20 个字符
Address	客户的地址，最长为 20 个字符

（续表）

列名	描述/数据类型
City	客户所在的城市，最长为 20 个字符
State	客户居住地所在的州，最长为 15 个字符
Country	客户居住的国家，最长为 20 个字符
Zip _ code	客户的邮政编码，最长为 15 个字符
Credit _ rating	客户的信誉等级，最长为 9 个字符
Sales _ rep _ id	客户的销售代表，最长为 3 个字符
Region _ id	客户居住的国家所在的区域，最长为 3 个字符
Comments	客户喜好的产品，最长为 255 个字符

表 1-6　S _ dept 表的属性

列名	描述/数据类型
Id	每个部门惟一的标识号，最长为 3 个字符
Name	部门名称，最长为 20 个字符
Region _ id	部门所在的区域 ID，最长为 3 个字符

表 1-7　S _ emp 表的属性

列名	描述/数据类型
Id	每个雇员惟一的标识号，最长为 3 个字符
Last _ name	雇员的姓氏，最长为 20 个字符
First _ name	雇员的名字，最长为 20 个字符
Userid	雇员的登录 ID，最长为 8 个字符
Start _ date	雇员开始在公司工作的日期，属于日期数据类型
Comments	雇员的有关信息，最长为 25 个字符
Manager _ id	雇员的经理的 ID，最长为 3 个字符
Title	雇员在公司里的职务，最长为 25 个字符
Dept _ id	雇员的部门 ID，最长为 3 个字符
Salary	雇员的工资，共 11 位数，包括 2 位小数
Commision _ pct	雇员赢得的佣金百分率，共 4 位数，包括 2 位小数

表 1-8　S _ region 表的属性

列名	描述/数据类型
Id	每个区域惟一的标识号，最长为 3 个字符
Name	每个区域惟一的名称，最长为 20 个字符

表 1-9 S_inventory 表的属性

列名	描述/数据类型
Product_id	产品惟一的标识号，最长为 7 个字符
Warehouse_id	存储产品的仓库 ID，最长为 7 个字符
Amount_in_stock	库存中每件产品的数量，最长为 9 个数字
Reorder_point	需要再订货的库存产品的最低数量，最长为 9 个数字
Max_in_stock	库存产品的最大值，最长为 9 个数字
Out_of_stock_explanation	产品无货的原因，最长为 255 个字符
Restock_date	产品再有货的日期，日期数据类型

表 1-10 S_item 表的属性

列名	描述/数据类型
ord_id	与项目有关的定单 ID，最长为 3 个字符
item_id	分配给每个项目的惟一的标识号，最长为 7 个字符
Product_id	与该项目有关的产品 ID，最长为 7 个字符
Price	该项目的价格，最长 11 位数，包括 2 位小数
Quantity	该项目的数量，最大为 9 位数
Quantity_shipped	已知产品的定单中该项目的发运数，最大为 9 位数

表 1-11 S_product 表的属性

列名	描述/数据类型
Id	产品惟一的标识号，最长为 7 个字符
Name	产品名，最长为 25 个字符
Short_desc	对产品的描述，最长为 255 个字符
Suggested_whlsl_price	产品的建议批发价，最长 11 位数字，包括 2 位小数
Whlsl_units	批发的产品单位，最长为 10 个字符

表 1-12 S_warehouse 表的属性

列名	描述/数据类型
Id	每个仓库的惟一标识号，最长为 7 个字符
Region_id	仓库所在的区域 ID，最长为 3 个字符
Address	仓库的地址，最长为 20 个字符
City	仓库所在的城市，最长为 20 个字符
State	仓库所在的州，最长为 15 个字符
Zip_code	仓库所在城市的邮政编码，最长为 15 个字符
Country	仓库所在的国家，最长为 20 个字符
Phone	仓库的电话号码，最长为 20 个字符
Manager_id	仓库经理的 ID，最长为 3 个字符

表 1-13　S_ord 表的属性

列名	描述/数据类型
Id	每个定单的惟一标识号，最长为 3 个字符
Customer_id	客户的惟一标识号，最长为 3 个字符
Date_ordered	定单的定货日期，日期数据类型
Date_shipped	定单的发运日期，日期数据类型
Sales_rep_id	负责定单的销售代表的惟一标识号，最长为 3 个字符
Total	定单的总金额，最长 11 位数，包括 2 位小数
Payment_type	支付方式，最长为 6 个字符
Order_filled	指出定单是否已经填写，最长为 1 个字符

1.15.2　运动用品数据库的参照完整性约束

表 1-14 列出了运动用品数据库表应满足的参照完整性约束。

表 1-14　运动用品数据库的完整性约束

表名	外码	参照属性/表
S_DEPT	region_id	表 S_REGION 的 ID 号
S_EMP	manager_id	表 S_EMP 的 ID 号
S_EMP	dept_id	表 S_DEPT 的 ID 号
S_EMP	title	表 S_TITLE 的标题
S_CUSTOMER	sales_rep_id	表 S_EMP 的 ID 号
S_CUSTOMER	region_id	表 S_REGION 的 ID 号
S_ORD	customer_id	表 S_CUSTOMER 的 ID 号
S_ORD	sales_rep_id	表 S_EMP 的 ID 号
S_ITEM	ord_id	表 S_ORD 的 ID 号
S_ITEM	prodct_id	表 S_PRODUCT 的 ID 号
S_WAREHOUSE	manager_i	表 S_EMP 的 ID 号
S_WAREHOUSE	region_id	表 S_REGION 的 ID 号
S_INVENTORY	product_id	表 S_PRODUCT 的 ID 号
S_INVENTORY	warehouse_id	表 S_WAREHOUSE 的 ID 号

1.15.3　运动用品数据库的附加约束

在运动用品数据库中还有以下约束：

- 客户的信誉等级只能取 EXCELLENT、GOOD 或 POOR 值。
- 在 S_DEPT 表中，name 及 region_id 的组合必须是惟一的。这一点确保了在地区中部门的惟一性。
- 任何雇员的佣金百分率必须为以下值之一：10、12.5、15、17.5 和 20。
- 在表 S_INVENTORY 中，属性 product_id 及 warehouse_id 的组合是惟一的。

1.16 表行的更新及删除

有时需对表中已存在的一行或多行进行修改，这是数据库的正常操作。记住，数据库不仅要记录现实世界的情况，而且要反映发生的变化。

1.16.1 UPDATE表命令

正如该命令的名称所暗示的一样，它的主要功能是更新表的某些行。该SQL命令用于更新单行的一个或多个列值时，其基本语法如下：

```
UPDATE table-name SET col-1 = new-val1 [...,...col-N = new-valN]
[WHERE condition];
```

这里，col-1……col-N表示列名，new-val1……new-valN表示将存储在相应列里的新值。WHERE子句允许我们改变被选中行的值。下面的例子说明如何使用这个命令。

例1.18

假定BJ Athletics已迁到位于佛罗里达州Melbourne的一个新地址，新的地址、邮政编码及电话号码分别是2905 Havens Avenue、32901和407-345-1265。请对运动用品数据库的S_CUSTOMER表中零售店BJ Athletic的信息进行更新。

为了在该数据库中改变此客户信息，必须对S_CUSTOMER表的相应行进行更新。用户对此表的属性可能不了解或未记住，他可以通过描述S_CUSTOMER表来确定表的结构。该表的结构如下：

```
DESC s_customer;
```

Name	Null?	Type
ID	NOT NULL	VARCHAR2(3)
NAME	NOT NULL	VARCHAR2(20)
PHONE	NOT NULL	VARCHAR2(20)
ADDRESS		VARCHAR2(20)
CITY		VARCHAR2(20)
STATE		VARCHAR2(15)
COUNTRY		VARCHAR2(20)
ZIP_CODE		VARCHAR2(15)
CREDIT_RATING		VARCHAR2(9)
SALES_REP_ID		VARCHAR2(3)
REGION_ID		VARCHAR2(3)
COMMENTS		VARCHAR2(255)

下面给出了反映 S_CUSTOMER 表中新变化的 UPDATE 命令。

```
UPDATE s_customer SET
phone='407-345-1265',
address = '2905 Fairway Dr.',
city = 'Melbourne',
state='FL',
zip_code='32901'
WHERE name = 'BJ Athletics';
```

用户可通过下列查询来证实改变是否正确：

```
SELECT phone, address, city, state, zip_code
FROM s_customer
WHERE name = 'BJ Athletics';
```

注意，没必要对任何其他属性进行查询，因为其他值并未改变。

在执行 UPDATE 命令时要小心，因为稍不留神就会出错。例如，如果用户发布以下 UPDATE 命令，则将会对 S_CUSTOMER 表的所有行进行错误更新

```
UPDATE S_customer SET
phone='407-345-1265',
address = '2905 Fairway Dr.',
city = 'Melbourne',
state='FL',
zip_code='32901';
```

失败的原因是查询所用的 WHERE 子句不合适。只要在执行 UPDATE 之前发布一个 SAVEPOINT A 命令，就很容易发现这个错误。一个简单的 ROLLBACK TO A 命令将使该表恢复到先前的状态。作者主张在对任何表进行更新之前，发布 SAVEPOINT 命令。

1.16.2　DELETE 及 TRUNCATE 命令

从数据库中删除一行或若干行是数据库正常操作的一部分。例如在运动用品数据库中，某个客户可能破产，某个雇员可能辞职。允许用户从表中删除行的 SQL 命令为 DELETE 命令。该命令可用于从表中删除满足指定条件的行，也可用于从一个特定的表中删除所有行。删除满足指定条件的行的命令语法如下：

```
DELETE FROM table-name WHERE condition;
```

删除一个表中所有行的 DELETE 命令的语法如下所示：

```
DELETE table-name
```

例 1.19

假设某一供应商不再生产 ID 为 32779 的产品，请从 S_INVENTORY 表中删除该元组。

假设我们知道的关于此产品的惟一信息是其 product_id，将它从数据库中删除的 DELETE 命令如下：

```
DELETE FROM s_inventory WHERE product_id='32779';
```

执行此命令时，用户必须十分小心，以防删除未在计划之内的行。例如，如果用户在上面的 DELETE 命令中省略了 WHERE 子句，S_INVENTORY 表的全部行均会被删除。

读者需记住，有时从一个表中删除某一指定行是不可能的，请看下例。

例 1.20

假设客户 One Sport 已停业，从 S_CUSTOMER 表中删除此客户。

假设我们知道的关于此客户的惟一信息是它的名称，从数据库中删除它的命令如下：

```
DELETE FROM s_customer WHERE name = 'One Sport';
```

然而，代替从 S_CUSTOMER 表中删除相应行的是，用户得到了一个违反完整性错误。产生这个错误的原因是用户欲删除的行被另一个表的外码所引用。此类错误的基本格式如下：

```
                ←——厂家内部ID错误编号
ERROR AAAA integrity constraint (table_name_column_name_fk)
violated - child record found
```

为了删除 One Sport 客户，用户需用 DELETE 或 UPDATE 命令删除或更新相应的外码。删除或添加 FK 是下章将讲解的话题。

将一个表的所有行删除的简单快捷方法是使用 TRUNCATE 命令。该命令的基本语法如下：

```
TRUNCATE TABLE table-name;
```

读者需记住，使用该命令是有一定限制条件的。首先，如果一个表是父表，且有参照完整性约束时，我们不能截断该表的行。其次，我们不能回退到 TRUNCATE 语句。下例将对限制条件进行讨论。

例 1.21

建立一系列运行脚本 SG_NO_CONSTRAINTS.sql[①]（见附录 D）的新表，并进行如下操作：

1. 给出存储在表 S_TITLE 中的职务。
2. 建立一个保留点，名为 before_update。
3. 在表 S_TITLE 中插入一个新的职务“仓库监督员”，不要提交该改变。
4. 给出存储到表 S_TITLE 中的职务。

① 除了所有的外码约束被删除外，该脚本与产生运动用品数据库的脚本一样。在运行该脚本时请确保将约束去除。

5. 回退到保留点 before _ update。

6. 给出存储到表 S _ TITLE 中的职务。

7. 建立保留点 before _ truncate。

8. 截断 S _ TITLE 表。

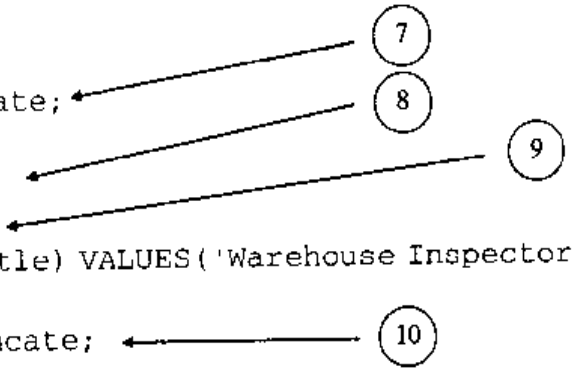
9. 在 S _ TITLE 表中插入新的职务“仓库监督员” 不要提交该改变

10. 回退到保留点 before _ truncate。

上面的一系列操作产生如下结果：

```
SQL> SELECT title FROM S_title;                                    ← (1)

TITLE
-------------------------
President
Sales Representative
Stock Clerk
VP, Administration
VP, Finance
VP, Operaticns
VP, Sales                                                          ← (2)
Warehouse Manager
8 rows selected.
SQL> SAVEPOINT before_update;                                      ← (3)
Savepoint created.
SQL> INSERT INTO S_title (title) VALUES('Warehouse Inspector');
1 row created.
SQL> SELECT title FROM S_title;                                    ← (4)
TITLE
-------------------------
President
Sales Representative
Stock Clerk
VP, Administration
VP, Finance
VP, Operations
VP, Sales
Warehouse Manager
Warehouse Inspector
9 rows selected.
SQL> ROLLBACK TO SAVEPOINT before_update;                          ← (5)
Rollback complete.
SQL> SELECT title FROM S_title;                                    ← (6)
TITLE
-------------------------
President
```

```
Sales Representative
Stock Clerk
VP, Administration
VP, Finance
VP, Operations
VP, Sales
Warehouse Manager
8 rows selected.
SQL> SAVEPOINT before_truncate;
Savepoint created.
SQL> TRUNCATE TABLE S_title;
Table truncated.
SQL> INSERT INTO S_title (title) VALUES('Warehouse Inspector');
1 row created.
SQL> ROLLBACK TO before_truncate;
rollback to before_truncate
*
ERROR at line 1:
ORA-01086: savepoint 'BEFORE_TRUNCATE' never established
```



注意，由 RDBMS 产生的错误表明：即使我们明确地建立了保留点 before_truncate，但它其实并不存在。产生该错误的原因在于 TRUNCATE TABLE 是一个 DDL 命令，不能产生回退信息。此外，像其他 DDL 命令一样，RDBMS 发出一个隐式 COMMIT 命令。

1.16.3 DROP TABLE 命令

有时需要从数据库中删除一个表及其中所有数据。在 SQL 命令中，DROP TABLE 命令可做到这一点。这个命令的基本格式如下：

DROP TABLE table-name [**CASCADE CONSTRAINTS**];

如果我们想要删除的某个表具有参照主码或惟一属性的参照完整性约束时，使用可选子句 CASCADE CONSTRAINTS。即当欲删除的某指定表是其他表的父表时，不使用 CASCADE CONSTRAINTS 子句就会出错。

通常在 CREATE TABLE 命令之前发出 DROP TABLE CASCADE CONSTRAINTS 命令，这样，当用户在自己模式中尝试建立一个与已存在的其他表具有相同名称的表时，可避免由 RDBMS 产生的任何错误。

问题与答案

1.1 利用下面给出的信息，写出建立 PROGRAMMER 表的 SQL 指令。假设该表的属性满足下面所示的条件。

属性名	描述/数据类型/约束
EmpNo	程序员的惟一 ID，最长为 3 个字符
Project	程序员所参与的项目，最长为 3 个字符
TaskNo	该项目相关的任务号，为数值列，最多 2 位数
Last _ Name	雇员的姓氏，最长为 25 个字符。需要
First _ Name	雇员的名字，最长为 25 个字符
Hire _ Date	雇员的雇佣日期，日期数据类型
Language	程序员使用的编程语言，最长为 15 个字符
Clearance	程序员的许可证的类别，最长为 25 个字符

属性 EmpNo 及 Project 可长至 3 个字符。属性 TaskNo 是一个数值列，最多可有 2 个数字。属性 Last _ Name、First _ Name 及 Clearance 可长至 25 个字符。属性 Language 可长至 15 个字符。给每一个程序员一个惟一的 EmpNo。需要提供姓氏。

EmpNo	Last Name	First Name	Hire Date	Project	Language	TaskNo.	Clearance
201	Campbell	John	1/1/95	NPR	VB	52	Secret
390	Bell	Randall	1/05/93	KCW	Java	11	Top Secret
789	Hixon	Richard	08/31/98	BNC	VB	11	Secret
134	McGurn	Robert	07/15/95	TIPPS	C + +	52	Secret
896	Sweet	Jan	06/15/97	KCW	Java	10	Top Secret
345	Rowlett	Sid	11/15/99	TIPPS	Java	52	
563	Reardon	Andy	08/15/94	NITTS	C + +	89	Confidential

相应的 CREATE TABLE 指令如下：

```
DROP TABLE programmer CASCADE CONSTRAINTS;
CREATE TABLE Programmer
  (EmpNo          Varchar2(3)        PRIMARY KEY,
  Last_Name       Varchar2(25)       NOT NULL,
  First_Name      Varchar2(25),
  Hire_Date       Date,
  Project         Varchar2(3),
  Language        Varchar2(15),
  TaskNo          Number(2),
  Clearance       Varchar2(25)
);
```

注意，我们在 CREATE TABLE 命令之前放上了 DROP TABLE programmer CASCADE CONSTRAINTS 命令，以删除用户可能拥有的名称相同的其他表，在每一个属性名内嵌入的空格被下划线代替，这点符合 1.7 节的命名指南。属性雇员号定义为此表的主码。属性 Last _ Name 被定义为 NOT NULL，因为它是一个不可或缺的属性。

1.2　创建一个脚本，用上述的数据填充 PROGRAMMER 表。用这些数据时，需要多少个

指令才能填充这个表?

需要写的脚本是一个单纯的文本文件，它包含了填充数据的所有指令。如果我们将脚本称为 programmer.sql，并且假设它在驱动器 A：中，则通过在 SQL 提示符下键入 @ "A：programmer.sql"就可以从 P08 运行它。由于 INSERT INTO 命令只允许一次加一行至表中，因此每行均需有 INSERT INTO 命令，共需要 7 个 INSERT INTO 命令来填充表。

注意，为避免发生错误，需要改变属性 HIRE_DATE 的数据。因为在缺省设置中，日期表示形式为："DD-MM-YY"，其中 DD 是某月的任一天(1～31)，MM 是月份前三位字母的缩写(JAN、FEB、DEC……)，YY 是年的后两位数字。这表明像 1/1/95 这个数字应改为 '1-JAN-95'，记住：要将日期置于单引号内。

观察雇员 Sid Rowlett 的属性 Clearance 所填的 NULL 值。填充 PROGRAMMER 表的脚本如下：

```
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('201','Campbell', 'John', '1-JAN-95','NPR', 'VB',52,
'Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('390', 'Bell', 'Randall', '1-MAY-93', 'KCW', 'Java', 11,
'Top Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values( 789', 'Hixon', 'Richard', '31-AUG-98', 'RNC', 'VB', 11,
'Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('134', 'McGurn','Robert', '15-JUL-95', 'TIP','C++',
52, 'Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('896','Sweet', 'Jan', '15-JUN-97', 'KCW','Java',10,
'Top Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('345','Rowlett', 'Sid', '15-NOV-99', 'TIP','Java',
52, NULL);
```

```
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('563', 'Reardon', 'Andy', '15-AUG-94', 'NIT', 'C++', 89,
'Confidential');
```

1.3　更新 PROGRAMMER 表，将 EmpNo = 345 的雇员出入许可证从 NULL 改为 Secret。使用雇员的编号作为行的限定词，更新 CLEARANCE 属性值的相应 SQL 命令如下：

```
UPDATE programmer
SET clearance = 'Secret'
WHERE empno = '345';
```

1.4　将下面元组插入到 PROGRAMMER 表中，然后显示所有雇员的姓氏、名字、出入许可证及雇用日期，并依据出入许可证对结果进行排序。

EmpNo	Last Name	First Name	Hire Date	Project	Language	TaskNo.	Clearance
597	Campbell	Alan	1/1/00	NPR	VB	52	Secret
390	Bell	Greg	1/1/00	KCW	Java	11	Top Secret

插入这两行的指令如下：

```
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values('198','Campbell', 'Alan', '1-JAN-00','NPR', 'VB',52,
'Secret');
INSERT INTO programmer
(EmpNo, Last_Name, First_Name, Hire_Date, Project, Language,
TaskNo, Clearance)
Values( 834',  Bell', 'Greg', '1-MAY-00', 'KCW', 'Java', 11,
'Top Secret');
```

显示所需信息的语句及产生的结果如下：

```
SELECT last_name, first_name, hire_date, clearance
FROM programmer
ORDER by clearance;
```

```
LAST_NAME      FIRST_NAME     HIRE_DATE      CLEARANCE
Reardon        Andy           15-AUG-94      Confidential
Campbell       John           01-JAN-95      Secret
Hixon          Richard        31-AUG-98      Secret
McGurn         Robert         15-JUL-95      Secret
Campbell       Alan           01-JAN-00      Secret
Rowlett        Sid            15-NOV-99      Secret
Bell           Randall        01-MAY-93      Top Secret
Bell           Greg           01-MAY-00      Top Secret
Sweet          Jan            15-JUN-97      Top Secret
```

1.5　写出反映雇员 EmpNo = 789 已辞职的 SQL 命令。

因为我们想从表中删除一行，因此需要使用 DELETE 命令，相应的 SQL 命令如下：

```
DELETE FROM programmer
WHERE empno='789';
```

1.6 运行SG.SQL脚本，刷新数据库内容，指出S_EMP表的度是多少？它的基数是多少？要回答这些问题应该使用哪些SQL指令？

度是在S_EMP表中属性或列的总数。基数是目前存储在表中的行的总数。为确定度，我们"描述"该表。也就是说，我们通过发出DESCRIBE s_emp命令来计算属性个数。

```
Name                          Null?          Type
----------------------------  -----------    ---------------
ID                            NOT NULL       VARCHAR2(3)
LAST_NAME                     NOT NULL       VARCHAR2(20)
FIRST_NAME                                   VARCHAR2(20)
USERID                        NOT NULL       VARCHAR2(8)
START_DATE                    NOT NULL       DATE
COMMENTS                                     VARCHAR2(255)
MANAGER_ID                                   VARCHAR2(3)
TITLE                                        VARCHAR2(25)
DEPT_ID                                      VARCHAR2(3)
SALARY                                       NUMBER(11,2)
COMMISSION_PCT                               NUMBER(4,2)
```

由于列出了11个属性，故S_EMP的度是11。

为了确定基数，我们需要给出一个SELECT语句，列出该表所有行。SELECT语句如下：

```
SELECT id, last_name, first_name, userid, start_date, comments,
       manager_id, title, dept_id, salary, commission_pct
FROM s_emp;
```

结果显示了25行，因此该表的基数为25。可观察到SELECT语句显示了所有行以后，Oracle给出了检索到的行的总数。Oracle中称之为反馈。缺省设置为检索到6行以上时才显示该反馈信息。在会话期间，可以使用以下命令在SQL提示符下随时改变该环境的设置：

```
SET feedback No_of_rows
```

此处No_of_rows是一个整数，表明我们在见到这个反馈信息之前需要检索行数的最小值。

1.7 在S_EMP表中，多少雇员在部门41工作？他们是谁？

欲回答此问题，需给出一个SELECT语句，以知道所有在部门41工作的雇员的姓名及ID。相应的WHERE子句是WHERE dept_id = '41'，完整的select语句如下：

```
SELECT id, last_name, first_name
FROM s_emp
WHERE dept_id ='41';
```

所得的关系如下：

```
ID      LAST_NAME          FIRST_NAME
2       Smith              Doris
6       Brown              Molly
15      Hardwick           Elaine
16      Brown              George
4 rows selected
```

1.8　是否有更简单的方法检索一个表中的所有列，而不必提及该表的全部属性?

对这个问题的回答是肯定的。在 SELECT 语句中有一个速写记号 * 可应付这类问题。但由于没有格式化表的列，因此出现在屏幕上的结果看起来会有些杂乱，该语句的通用格式如下：

```
SELECT *
FROM table-name
[WHERE condition]
[ORDER BY];
```

1.9　将 1.7 题所得的结果按雇员姓氏的字母顺序排列。

由于我们欲对 S_EMP 表中满足 WHERE 条件的行进行排序，故需用 SELECT 语句的 ORDER BY 子句。相应的 SELECT 语句如下：

```
SELECT id, last_name, first_name
FROM s_emp
WHERE dept_id ='41'
ORDER by last_name;
```

所得的结果表如下：

```
ID      LAST_NAME          FIRST_NAME
6       Brown              Molly
16      Brown              George
15      Hardwick           Elaine
2       Smith              Doris
```

1.10　上题的结果是根据雇员姓氏排序的，但是我们可以注意到，对于具有相同姓氏的雇员，并未进一步按他们的名字进行排序。举例来说，Brown George 本应该在 Brown Molly 之前。能做到这一点吗?

是的，能做到，做到这一点的指令如下：

```
SELECT id, last_name, first_name
FROM s_emp
WHERE dept_id ='41'
ORDER by last_name, first_name;
```

结果表的行首先以姓氏排序，对姓氏相同的人再依据名字排序。要注意 ORDER BY 子句中的属性顺序，这一点非常重要

可以看出，我们已将 first_name 列加到 ORDER BY 子句中，且列名之间以逗号隔开。注意，属性的顺序是至关重要的。在本例中，我们首先以姓氏排列，对姓氏相同的人再依据名字排序。所得的结果表如下所示：

```
ID      LAST_NAME      FIRST_NAME
16      Brown          George
6       Brown          Molly
15      Hardwick       Elaine
2       Smith          Doris
```

这两行首先按姓氏排序，对姓氏相同的人再按名字排序

如果存在另一列如 middle_name，我们希望按顺序排列时，ORDER BY 就可以这样写：

```
ORDER by last_name, first_name, middle_name;
```

这样所得的结果表首先以姓氏排序，对姓氏相同的人再依据名字排序，对姓氏及名字相同的人再依据中间名排序。

1.11 写一个脚本，为零售店 Waves-R-Us 建一个产品表 Product。该表属性如下所示。如果需要的话可使用命令约束，并证实建立的表是正确的。

属性名	说明
ID	惟一的产品标识，最长为5个字符，需要
NAME	产品名，最长为25个字符，需要
DISCOUNT PERCENTAGE	对优选客户的价格折扣百分率，最多1位数
PRICE	产品的零售价，总共6位数字，其中带2位小数

ID	NAME	PRICE	DISCOUNT_PRCNTG
111	Surfing board	225.95	5
200	Ear plugs	10	2
345	Goggles	35	1

建立这个表的 SQL 指令如下。对属性 ID 及 NAME 的约束依据表 1.2 的惯例进行。

```
CREATE TABLE Product

(
 id   Varchar2(5)    CONSTRAINT Products_id_pk      PRIMARY KEY,
 Name Varchar2(25)   CONSTRAINT Products_name_nn    NOT NULL,
 Discount_prcntg     Number(1),
 Price               Number(6,2)
);
```

为了证实所建立的表是正确的，需要使用 DESCRIBE product 命令确定它的结构。

```
DESCRIBE product;

 Name                          Null?          Type
 ----  ----------------        ----------     ----------------
 ID                            NOT NULL       VARCHAR2(5)
 NAME                          NOT NULL       VARCHAR2(25)
 DISCOUNT_PRCNTG                              NUMBER(1)
 PRICE                                        NUMBER(6,2)
```

1.12 DESCRIBE product 命令的输出展示了表的结构，但它并未显示属性的约束。如何知道是否已正确命名了约束呢?

为了知道是否已为 Product 表正确命名了约束，需要查阅 USER_CONSTRAINTS 数据

字典视图[1]，该视图正如它的名字指出的那样，包含了指定表已定义的全部约束信息。对某一指定表已定义的所有约束名称及类型的通用查询格式如下：

```
SELECT constraint_name, constraint_type

FROM user_constraints                          表名必须用大写字母

WHERE table_name = 'TABLE-ON-WHICH-CONSTRAINTS-ARE DEFINED';
```

Product 表的相应 SELECT 命令如下：

```
SELECT constraint_name, constraint_type

FROM user_constraints

WHERE table_name='PRODUCT';
```

注意在 WHERE 子句中，表名必须大写，且置于单引号内。因为在数据字典中，表名以大写字母存储。该查询结果如下：

```
CONSTRAINT_NAME                C  TABLE_NAME
------------------------------ -  ------------------------------
PRODUCTS_NAME_NN               C  PRODUCT
PRODUCTS_ID_PK                 P  PRODUCT
```

列 constraint_type 以列 C 表示，该列的值及它们的含义如下：

C 表明约束属于 CHECK 类型，在这种情况下，RDBMS 对非空值进行检查。

P 表明约束与主码结合。

U 表明约束与惟一性约束结合。

R 表明约束与外码结合。

1.13　USER_CONSTRAINTS 视图的列是什么？每一列表示什么意思？

使用 DESCRIBE USER_CONSTRAINTS 命令可获得 USER_CONSTRAINTS 视图的结构。这个视图的列的一部分如下所示：

```
Name                           Null?    Type
------------------------------ -------- ----------
OWNER                          NOT NULL VARCHAR2(30)
CONSTRAINT_NAME                NOT NULL VARCHAR2(30)
CONSTRAINT_TYPE                         VARCHAR2(1)
TABLE_NAME                     NOT NULL VARCHAR2(30)
SEARCH_CONDITION                        LONG
R_OWNER                                 VARCHAR2(30)
R_CONSTRAINT_NAME                       VARCHAR2(30)
DELETE_RULE                             VARCHAR2(9)
STATUS                                  VARCHAR2(8)
LAST_CHANGE                             DATE
```

① 将在第 8 章讨论视图。

USER_CONSTRAINTS 视图所选各列的含义如下：

列	说明
OWNER	定义(拥有)该约束的用户
CONSTRAINT_NAME	约束名
CONSTRAINT_TYPE	约束类型
TABLE_NAME	定义约束的表名
SEARCH_CONDITION	CHECK 约束查找条件的文本
R_OWNER	在 FK 中被参照的、建立了父表(属主)的用户
R_CONSTRAINT_NAME	在父表中被参照的 PK 名或 UNIQUE 属性名
DELETE_RULE	针对参照已被删除的 PRIMARY 或 UNIQUE 码的 FK 所采取的行动；被支持的行动为 CASCADE 和 NO ACTION
LAST_CHANGE	最后一次修改约束的日期

1.14 对于一个给定表，如何找出定义了约束的列名称？

为回答上述问题，使用 USER_CONS_COLUMNS 数据字典视图。该视图的结构如下：

```
SQL> DESC USER_CONS_COLUMNS;

 Name                           Null?    Type
 ------------------------------ -------- ----------------
 OWNER                          NOT NULL VARCHAR2(30)
 CONSTRAINT_NAME                NOT NULL VARCHAR2(30)
 TABLE_NAME                     NOT NULL VARCHAR2(30)
 COLUMN_NAME                             VARCHAR2(4000)
 POSITION                                NUMBER
```

PRODUCTS 表相应的 SELECT 语句及其执行结果如下：

```
SELECT column_name, constraint_name
FROM user_cons_columns
WHERE table_name= 'PRODUCTS';

CONSTRAINT_NAME                COLUMN_NAME
------------------------------ --------------------
PRODUCTS_ID_PK                 ID
PRODUCTS_NAME_NN               NAME
```

1.15 作为一位用户，如何找到自己已建立的表的总个数或自己能存取的表的个数？

欲回答此问题，查询数据字典视图 USER_TABLES。用户可存取的表名或已建立的

表名可通过以下查询给出：

```
SELECT table_name
FROM user_tables;
```

1.16　根据 S_EMP 表，列出所有雇员的姓氏、ID 及职务。

相应的 SQL 查询是：

```
SELECT last_name, userid, title
FROM s_emp;
```

该查询的部分结果如下所示。注意该查询结果已被格式化了。

LAST_NAME	USERID	TITLE
Martin	martincu	President
Smith	smithdj	VP, Operations
Norton	nortonma	VP, Sales
Quentin	quentiml	VP, Finance
Roper	roperjm	VP, Administration
Brown	brownmr	Warehouse Manager
Hawkins	hawkinrt	Warehouse Manager
Burns	burnsba	Warehouse Manager
Catskill	catskiaw	Warehouse Manager
Jackson	jacksomt	Warehouse Manager
Henderson	hendercs	Sales Representative
Gilson	gilsonsj	Sales Representative
Sanders	sanderjk	Sales Representative
Dameron	dameroap	Sales Representative
Hardwick	hardwiem	Stock Clerk
Brown	browngw	Stock Clerk
Washington	washintl	Stock Clerk
Patterson	patterdv	Stock Clerk
Bell	bellag	Stock Clerk
Cantos	gantosej	Stock Clerk
Stephenson	stephebs	Stock Clerk
Chester	chesteek	Stock Clerk
Pearl	pearlrg	Stock Clerk
Dancer	dancerbw	Stock Clerk
Schmitt	schmitss	Stock Clerk

25 rows selected.

1.17　按职务的字母以降序方式显示上题的查询结果。

相应的 SQL 查询如下：

```
SELECT last_name, userid, title
FROM s_emp
ORDER BY title DESC;
```

该查询的结果如下所示：

```
LAST_NAME                 USERID       TITLE
------------------------- ------------ -------------------------
Brown                     brownmr      Warehouse Manager
Hawkins                   hawkinrt     Warehouse Manager
Burns                     burnsba      Warehouse Manager
Jackson                   jacksomt     Warehouse Manager
Catskill                  catskiaw     Warehouse Manager
Norton                    nortonma     VP, Sales
Smith                     smithdj      VP, Operations
Quentin                   quentiml     VP, Finance
Roper                     roperjm      VP, Administration
Hardwick                  hardwiem     Stock Clerk
Brown                     browngw      Stock Clerk
Bell                      bellag       Stock Clerk
Patterson                 patterdv     Stock Clerk
Washington                washintl     Stock Clerk
Gantos                    gantosej     Stock Clerk
Chester                   chesteek     Stock Clerk
Dancer                    dancerbw     Stock Clerk
Pearl                     pearlrg      Stock Clerk
Schmitt                   schmitss     Stock Clerk
Stephenson                stephebs     Stock Clerk
Henderson                 hendercs     Sales Representative
Sanders                   sanderjk     Sales Representative
Dameron                   dameroap     Sales Representative
Gilson                    gilsonsj     Sales Representative
Martin                    martincu     President
25 rows selected.
```

1.18 按以下要求对 S_EMP 表中所有雇员的职务、姓氏及部门排序。要求不同的职务按字母顺序排列，对具有相同职务的雇员，按照其姓氏的降序排列

相应的 SQL 查询如下：

```
SELECT title, last_name, dept_id
FROM s_emp
ORDER BY title, last_name DESC;
```

该查询的结果如下：

```
TITLE                     LAST_NAME                 DEP
------------------------- ------------------------- ---
President                 Martin                    50
Sales Representative      Sanders                   33
Sales Representative      Henderson                 31
Sales Representative      Gilson                    32
Sales Representative      Dameron                   35
Stock Clerk               Washington                42
```

```
Stock Clerk                Stephenson          45
Stock Clerk                Schmitt             45
Stock Clerk                Pearl               34
Stock Clerk                Patterson           42
Stock Clerk                Hardwick            41
Stock Clerk                Gantos              44
Stock Clerk                Dancer              45
Stock Clerk                Chester             44
Stock Clerk                Brown               41
Stock Clerk                Bell                43
VP, Administration         Roper               50
VP, Finance                Quentin             10
VP, Operations             Smith               41
VP, Sales                  Norton              31
Warehouse Manager          Jackson             45
Warehouse Manager          Hawkins             42
Warehouse Manager          Catskill            44
Warehouse Manager          Burns               43
Warehouse Manager          Brown               41
25 rows selected.
```

1.19　显示在部门 41 工作的所有雇员姓氏及工资。

相应的 SQL 查询如下：

```
SELECT last_name, salary
FROM s_emp
WHERE dept_id = '41';
```

该查询的结果如下：

```
LAST_NAME                  SALARY
Smith                      2450
Brown                      1600
Hardwick                   1400
Brown                       940
4 rows selected.
```

1.20　给在部门 41 工作的所有雇员提高工资。对结果加以验证。万一用户出错，确保所有的变化均能取消。如果操作有误，用户如何取消这些变化。

为了增加在部门 41 工作的雇员的工资，有必要更新 S_EMP 表的相应行。但在对表作出任何改变之前，首先要定义一个保留点，这种方法可使我们以后能取消这种改变。

```
SAVEPOINT before_update;
```

目前部门 41 所有雇员的工资如下：

```
LAST_NAME                  SALARY
-------------------------  --------
Smith                      2450
Brown                      1600
Hardwick                   1400
Brown                       940
4 rows selected.
```

更新部门 41 雇员工资的 SQL 命令如下：

```
UPDATE S_emp
SET salary = salary + 1000
WHERE dept_id = '41';
```

通过以下查询可对改变是否正确加以验证：

```
SELECT last_name, salary
FROM s_emp
WHERE dept_id = '41';
```

此查询结果如下：

```
LAST_NAME              SALARY
Smith                  3450
Brown                  2600
Hardwick               2400
Brown                  1940
4 rows selected.
```

如果更新不正确，用户可通过以下命令取消这些变化。

```
ROLLBACK TO SAVEPOINT before_update;
```

1.21 在 Oracle 中，如何将交互式会话中键入的命令保存至一个文件？

为了保存一个交互式会话命令到一个指定文件中，可使用以下 SQL * Plus 命令：

```
SAVE filename
```

为了在同一个文件中积累多个交互式命令，可使用以下 SQL * Plus 命令。在所有情况下，可将完整的路径描述放在文件名之前。

```
SAVE filename REPLACE
```

如果用户没指出路径名，文件将被保存在一个缺省目录里。如果用户没指出文件扩展名，系统将使用缺省扩展名 .SQL

1.22 如何执行以前用 SAVE 命令保存的查询？

为了执行存储在一个文件里的命令，可使用以下 SQL * Plus 命令：

```
START filename
```

如果文件的路径名及扩展名不是缺省值，则文件名前面必须有完整的路径名及文件扩展名。

1.23 运行脚本来刷新问题与答案 1.2 的 PROGRAMMER 表。假设由于项目 KCW 的完成，所有在该项目工作的雇员均被调回公司总部。要求从 PROGRAMMER 表中删除这些雇员信息。

将雇员从 PROGRAMMER 表中删除的 SQL 命令如下：

```
DELETE FROM programmer WHERE project = 'KCW';
```

1.24 刷新 PROGRAMMER 表，并显示有 Secret 许可证的程序员姓名及项目。

产生这个信息的查询如下：

```
SELECT last_name, project, clearance
FROM programmer
WHERE clearance = 'Secret';
```

该查询结果如下：

```
LAST_NAME                 PRO   CLEARANCE
------------------------- ----  ----------------
Campbell                  NPR   Secret
Hixor                     RNC   Secret
McGurn                    TIP   Secret
3 rows selected.
```

1.25 由于工作需要，PROGRAMMER 表中所有具有 Secret 许可证的雇员都被授予 Top Secret 许可证。写出反映许可证升级的相应 SQL 命令。

相应的 SQL 命令如下：

```
UPDATE programmer SET clearance = 'Top Secret'
WHERE clearance = 'Secret';
```

1.26 用三个不同精度的数值属性建立表 TOOL。

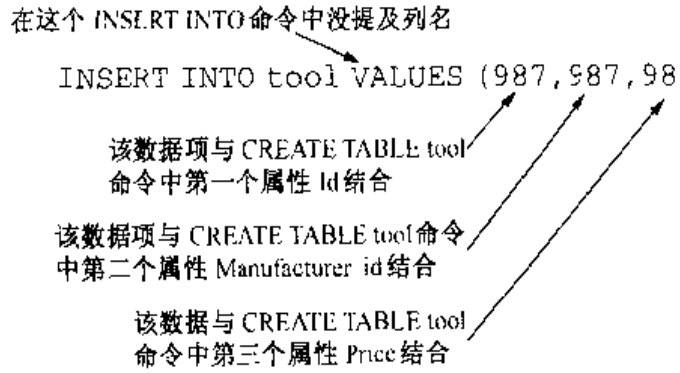
```
CREATE TABLE tool
(
 Id                  NUMBER,
 Manufacturer_id     NUMBER (5),
 Price               NUMBER (5,2)
);
```

作为执行 INSERT INTO 命令的结果，下一页显示了在该表的每一列中存储的值。属性 Id 允许在小数点左边或右边输入任何个数的数字，属性 Manufacturer_id 允许在小数点左边不多于 5 位数，如果任何数字被插入小数点的右边，该数字将被四舍五入。属性 Price 允许总数不超过 5 位数，其中 2 位数在小数点右边，其余 3 位在小数点左边。INSERT INTO 命令的结果如下：

INSERT INTO commands	Id	Manufacturer_id	Price
INSERT INTO tool (Id, Manufacturer_id, Price) VALUES (987,987,987);	987	987	987
INSERT INTO tool (Id, Manufacturer_id, Price) VALUES (987.222,987.225,987.225);	987.222	987	987.23
INSERT INTO tool (Id, Manufacturer_id, Price) VALUES (98.55,98.55,98.55)	98.55	99	98.55
INSERT INTO tool (Id, Manufacturer_id, Price) VALUES (98765,98765,98765)			因为此属性值位数过多，插入失败

1.27 当填充表的内容时，总需要在 INSERT INTO 子句里写出列名吗？

不是。如果写在 VALUES 子句中的数据项的顺序与 CREATE TABLE tool 命令中表的属性顺序相同，则不用在 INSERT INTO 子句中写出列名。我们用上一题的三个 INSERT INTO 命令中的第一个命令进行说明。注意，在 CREATE TABLE tool 命令中，属性是按以下顺序给出的：Id、Manufacturer _ id 及 Price 。



同样我们可以写出第二个及第三个 INSERT INTO 语句，如下所示：

```
INSERT INTO tool VALUES(987.222,987.225,987.225);

INSERT INTO tool VALUES(98.55, 98.55, 98.55);
```

在使用这种格式的 INSERT INTO 命令之前，用户必须保证以正确的顺序列出数据项。

补 充 题

1.28 写出建立一个包含样本天气信息的五栏表的 SQL 语句。该表的属性需满足以下约束：1）属性 City name 必须存在，可多达 13 个字符。2）属性 Sample _ Date 是日期类型，也必须有 3）正午和午夜的温度最多可有 3 位数字，包含 1 位小数。4）属性 Precipitation 最多可有 5 位数，包含 1 位小数。

1.29 建立一个名称为 STAFF 的表，其属性和约束如下。请对每个属性使用最合适的数据类型，并给所有必不可少的约束命名。使用你认为必要的属性长度。

属性	约束
First Name	需要
Last Name	需要
Title	可多达 15 个字符长
Id	码，可多达 5 个字符长
Salary	数值(最多 6 位数字，包含 2 位小数)
Department	需要，可多达 10 个字符长

1.30 从一个表中删除元组时，DELETE 及 DROP TABLE 命令似乎有同样的效果。当从一个表中删除元组时，DELETE 及 DROP TABLE 命令有区别吗？

1.31 如果存在下面表的定义，那么在执行了以下的 INSERT INTO 命令后，在每一个属性

中存储了什么值?

```
CREATE TABLE trynum

(
 first number,
 second number (2),
 third number (2,2)
);

INSERT INTO trynum (first, second, third)
            VALUES (.00005, .00005, .00005);
INSERT INTO trynum (first, second, third)
            VALUES (1.9, 1.9, 1.9);
INSERT INTO trynum (first, second, third)
            VALUES (10, 10, 10);
```

1.32　运行脚本 World _ Cities.sql，按照以下顺序显示属性：continent、country 及 city，确保洲名按字母逆向顺序列出，位于同一洲的国家名按字母顺序列出，位于同一国家的城市按字母逆向顺序列出。

1.33　在 World _ City 表中，显示不在亚洲的城市名。按洲的顺序输出结果，对同一洲，按国家名称的顺序给出，对同一国家，按城市名的顺序给出。

1.34　在 World _ Cities 表中，显示全部欧洲城市的名称及经度和纬度信息。按字母逆向顺序输出城市名。

1.35　运行脚本 SG.sql，显示在 1990 年 3 月 8 日聘用的所有雇员的姓氏、用户 ID 及开始工作的日期.

1.36　显示不在部门 44 工作的所有雇员的姓氏、名字及部门 ID。

1.37　描述数据字典视图 USER _ OBJECTS，显示用户拥有的所有表名

1.38　显示所有库存管理员的姓氏、名字及工资。结果以工资的升序给出。

1.39　显示在 1991 年 8 月 31 日后聘用的所有雇员的姓氏、名字及聘用日期。

1.40　显示工资少于 1200 美元的所有雇员的姓氏、名字及工资。

1.41　用 DROP TABLE 命令删除表 S _ EMP 会出现什么情况，为什么?

补充题答案

1.28

```
CREATE TABLE climate (
City              Varchar2(13) not null,
Sample_Date       DATE not null,
Noon              Number(3,1),
Midnight          Number(3,1),
Precipitation     Number(5,1)
);
```

1.29

```
DROP TABLE staff CASCADE CONSTRAINTS;
CREATE TABLE staff (
First_Name  Varchar2(20)  CONSTRAINT  staff_first_name_nn
                          NOT NULL,
Last_Name   Varchar2(20)  CONSTRAINT  staff_last_name_nn
                          NOT NULL,
Title       Varchar2(15),
Id          Varchar2(5)   CONSTRAINT  staff_id_pk PRIMARY KEY,
Salary      Number(6,2),
Department  Varchar2(10)  staff_department_nn NOT NULL);
```

1.30 DELETE 及 DROP TABLE 命令是有区别的。DELETE FROM 命令可根据 WHERE 条件，从一个表中删除一行或多行，但这个指令保留了表的结构。DROP TABLE 命令将表及其内容全部删除。如果 DROP TABLE 命令被成功执行的话，该表在数据库中不再存在。

1.31 在第一个 INSERT INTO 命令后，在表的属性中存储的值如下所示。注意，SECOND 属性及 THIRD 属性值都为 0，这是因为它们的小数点后数字分别要求为 0 位数及 2 位数，经过四舍五入均得到 0 值。

```
FIRST       SECOND      THIRD
----------  ----------  ----------
.00005      0           0
```

第二个 INSERT INTO 命令产生一个错误，这是由于属性 THIRD 的精度造成的。如果允许 2 个数字，并且小数点后要有 2 位数，小数点左边有任何值都会比要求的值大。显示的错误原因是“值大于该列允许的指定精度”。

第三个 INSERT INTO 命令也失败了。RDBMS 产生了一个与前一条 INSERT INTO 命令同样的错误。

1.32

```
SELECT continent, country, city
FROM World_Cities
ORDER BY continent DESC, country, city DESC;
```

1.33

```
SELECT continent, country, city
FROM World_Cities
WHERE Continent <> 'ASIA'
ORDER BY Continent, country, city;
```

1.34

```
SELECT City, Latitude, NorthSouth, Longitude, EastWest
FROM World_cities
WHERE Continent = 'EUROPE'
ORDER BY City DESC;
```

1.35

```
SELECT last_name, userid, start_date
FROM s_emp
WHERE start_date = '08-MAR-90';
```

1.36

```
SELECT last_name,first_name,dept_id
FROM S_emp
WHERE dept_id <> '44';
```

1.37

```
DESCRIBE USER_OBJECTS;
```

结果随着用户建立的表的不同而不同。回答该问题的查询如下：

```
SELECT object_name
FROM user_objects
WHERE object_type = 'TABLE';
```

1.38

```
SELECT last_name, first_name, salary
FROM s_emp
WHERE title = 'Stock Clerk'
ORDER by salary;
```

1.39

```
SELECT last_name, first_name, start_date
FROM s_emp
WHERE start_date > '31-AUG-91';
```

1.40

```
SELECT last_name,last_name,salary
FROM s_emp
WHERE salary < 1200;
```

1.41　表 S_EMP 不能被删除，因为它的一些属性被另外一些表的 FK 参照。显示的错误原因是“表中的惟一性/主码被外码参照”。要删除该表，用户需使用命令 DROP TABLE s_emp CASCADE CONSTRAINTS。

第 2 章　SQL 中关系运算符的执行

上一章对 SQL 语言的一些基本操作特点进行了讲解。在本章的余下部分，我们将集中学习在第 1 章第 1.5 节阐明的理论关系运算符的执行。此外，我们还对 CREATE TABLE 及 SELECT 命令的其他特点进行拓展，并对能够加强完整性约束及操纵表结构的 DDL 语句进行说明。

2.1　选择运算符的执行

对一个给定关系 r 进行选择运算可产生一个新的关系，新关系具有与 r 同样的属性，并且其中的行是在 r 中满足指定条件的行。正如其他所有的关系运算符一样，这个运算符也是使用 SELECT 语句来执行的。在本节中，我们将考虑该语句的一些变化，阐明在 SQL 中选择运算符的执行。

例 2.1

在运动用品数据库中，显示 S_DEPT 表中 Operations 部门的全部信息。

欲回答这个查询，我们需要显示 S_DEPT 表中满足条件 NAME = 'Operations' 的各行的全部列。显示所需要信息的 SELECT 语句如下：

```
SELECT regior_id, name, id
FROM S_dept
WHERE NAME = 'Operations';
REG NAME                          ID
--- ---------------------- ---
1   Operations                    41
2   Operations                    42
3   Operations                    43
4   Operations                    44
5   Operations                    45
5 rows selected.
```

这个运算结果所显示的列的标题用大写字母给出，并与 SELECT 子句中的顺序一样。该结果有一个令人感兴趣的地方，那就是列标题 region_id 的显示形式。即使在 SELECT 命令中使用的是全称的列名，在结果中，还是只显示了该列的头三个字符。这是因为在建立 S_REGION表时，属性 region_id 被定义为 VARCHAR2(3)。Oracle RDBMS 以缺省方式显示与列定义具有相等字符数的列的内容。我们可通过对表 S_DEPT 的描述来证实这一点（见下面）。列 ID 及列 NAME 的名称被完全给出，因为它们分别被定义为 VARCHAR2(3) 和

VARCHAR2(20)，这些列定义的宽度足以使全部列名以标题形式显示出来。

```
Name                    Null?           Type
ID                      NOT NULL        VARCHAR2(3)
NAME                    NOT NULL        VARCHAR2(20)
REGION_ID                               VARCHAR2(3)
```

显示一个表的全部内容的另一种方法已经在第 1 章提及(见"问题与答案 1.8")。在这个 SELECT 语句的变化中，我们用一个 * 号来代替全部的列名。下面的例子对此进行了说明。

例 2.2

要求在不明确给出列名的情况下，显示 SC 数据库中 S_DEPT 表 Operations 部门的所有信息。

使用 * 号来替代所有的列名，获得该信息的查询如下所示：

```
SELECT *
FROM s_dept
WHERE NAME = 'Operations';
ID  NAME                      REG
--- ------------------------- ---
41  Operations                1
42  Operations                2
43  Operations                3
44  Operations                4
45  Operations                5
5  rows selected.
```

可以观察到在这个查询显示的结果中，与上例中所得到的显示结果有一点细微的差别，特别是给出的各个列名的顺序不同。当我们使用一个星号来显示表的各列时，Oracle 将以在 FROM 子句中提及的表，在该表的 CREATE TABLE 命令中定义的相同的顺序显示这些列。可通过观察 S_DEPT 表的结构来证实这一点。按照这个原则，当 S_DEPT 表建立后，第一列定义为 ID，然后是 NAME，最后是 REGION_ID，该 SELECT 语句正是以这个顺序从左到右显示了表 S_DEPT 的各列。

允许我们显示一个表的每一列及每一行的选择运算符的通则如下：

```
SELECT *
FROM table-name;
```

该语句相当于用一个未指定 WHERE 条件进行的选择运算。FROM 子句中提到的表的每一行都满足这个未指定的条件。下面对这一点进行说明。

例 2.3

显示 S_DEPT 表的全部内容。

该查询及部分输出结果如下：

```
SELECT *
FROM s_dept;
ID  NAME                 REG
--- -------------------- ---
10  Finance              1
31  Sales                1
.
.
12 rows selected.
```

可以看出，* 号的使用使查询的各列以与 CREATE TABLE s_dept 命令中定义的相同的顺序显示。

2.2 用别名控制列标题

到目前为止，在讲解的所有查询中，我们允许 RDBMS 用缺省值显示数据。例如，在例 2.1 中，属性 region_id 的标题仅以三个字符显示。让 RDBMS 使用缺省值显示结果可能导致列标题无任何意义。有时我们想控制列标题的显示形式。例如，当不想让用户知道查询的某个表的列名的时候。在这种情况下，我们想使用替代名来代替列标题，这个替代名被称为列别名。

在 SELECT 语句的 SELECT 子句中，列标题被改变了。建立列别名的两种基本方法如下：第一，在列名后接关键字 AS，并将列别名置于双引号内。第二，在列名后，直接接上括在双引号内的列别名，中间不用关键字 AS。这两个建立列别名的方法是等效的，但使用关键字 AS 来建立列别名似乎更明确，而且使 SELECT 语句易于阅读。在本书中，这两种方法均被用于建立列别名。即使别名仅仅是一个单词，我们也提倡将别名置于双引号内。如果别名包括大写及小写字母，由被空格隔开的多个单词组成，或包含一些特殊字符，如 $ 或 #，则别名必须置于双引号内。读者必须记住：用户使用别名并不会改变一个表的列名，别名仅充当用于显示的标题。

当我们不想理睬 Oracle "显示字符数不应多于列定义中的字符数" 的约束时，可以用 SQL * Plus 的 COLUMN 命令设计列。下例讲述了别名及 COLUMN 命令的使用。

例 2.4

使用 Last Name、First Name 和 Salary 作为列标题，显示运动用品数据库中所有雇员的姓氏、名字及工资。

由于新标题 Last Name 及 First Name 是由一个空格隔开的两个单词，因此必须将这些别名置于双引号内。标题 Salary 也必须置于双引号内，因为它有大小写混合的情况。SELECT 语句及它产生的标题如下。注意，我们只显示了该查询结果的头两行。

```
SELECT last_name AS "Last Name", first_name AS "First
Name", salary AS "Salary"
FROM s_emp;
```

```
Last Name                 First Name              Salary
------------------------- ----------------------- ---------
Martin                    Carmen                     4500
Smith                     Doris                      2450
.
.
.
25 rows selected.
```

SQL * Plus 的 COLUMN 命令，不仅允许我们不用 RDBMS 的缺省值来显示标题，而且允许我们格式化数据的显示。在本书中我们对这个命令使用以下语法：

COLUMN column-name HEADING new-heading [FORMAT format-mask]

选项 format-mask(格式掩码)详细说明了列数据的显示格式。表 2-1 给出了本书中使用的一些格式掩码及它们对被显示数据的影响：

表 2-1　格式掩码

掩码	效果	例子	说明
An	用 n 个字符宽显示一列	A10	显示 10 个字符，更长列的多余字符被截掉
9	显示一位有效数字	999	显示 3 位有效数字，最前面的 0 显示为空格
0	在这个位置显示以 0 开头或一个零值	0999	如果所有 9 的数字为 0，就在这个位置显示一个 0 或在这个位置以 0 开头
,	在这个位置显示一个逗号	9,999	将值 2456 显示为 2,456
.	在这个位置显示一个分位符	9,999.99	将值 2456 显示为 2,456.00
$	显示美元符号	$999.99	将值 25 显示为 $25.00

下面的例子对这个格式化命令的使用进行了说明。

例 2.5

使用 Region Number、Department Name 和 Department Id 作为标题，显示运动用品数据库中所有 Operations 部门的区域、部门名及其 ID。

本查询需要显示的信息与例 2.1 几乎一样，但 COLUMN 命令的使用使标题的显示稍有不同。该查询所显示的标题及结果的头两行如下所示：

```
COLUMN region_id HEADING "Region|Number" FORMAT A10
COLUMN id HEADING "Department|Id" FORMAT A10
SELECT region_id, name AS "Department Name", id
FROM S_dept
WHERE NAME = 'Operations';
Region                          Department
Number     Department Name      Id
---------- -------------------- ----------
1          Operations           41
2          Operations           42
.
.
.
5 rows selected.
```

注意标题分隔字符 | 的使用，它使一些标题分为两排。看一下如何将 COLUMN 命令及一个列别名联合起来使用，以改变标题。在这个例子中特别需要注意的是，我们必须使用 COLUMN 命令来产生这些标题。如果不使用 COLUMN 命令，RDBMS 就会使用定义的列宽度来显示列名或其别名，标题会呈现如例 2.1 所示的截短形式。

读者必须注意：无论何时使用 COLUMN 命令对指定的列标题进行格式化，在一个新的 COLUMN 命令改变该标题或在该列标题被 CLEAR 之前，该标题的格式始终有效。在 SELECT 子句中，用 AS 定义的别名是暂时的，仅为显示之用，在 SELECT 语句被执行后，该列别名不再存在。

欲清除某一列的标题，可使用以下 SQL * Plus 命令：

```
COLUMN column-name CLEAR
```

欲清除所有的列标题，可使用以下 SQL * Plus 命令：

```
CLEAR COLUMNS
```

欲显示目前所有列的设定，可使用以下 SQL * Plus 命令：

```
COLUMN
```

2.3 投影运算符的执行

当投影运算符用于关系 r 及其属性的集合 X 时，会产生一个没有重复行，且属性是 X 的元素的新关系。在 SQL 中，这个运算符的执行通过使用 SELECT 命令中 SELECT 子句的 DISTINCT 选项来进行(见图 2-1)。DISTINCT 选项的使用阻止了重复行的出现。新关系的属性是 SELECT 子句以通常方式列出的参与投影的表的属性。注意，在 SELECT 命令的语法中用 ALL 表示 SELECT 子句的缺省值。这就是说，当用户不明确地指明时，SELECT 命令将显示所有的行，包括一些重复行。

```
SELECT [ALL | DISTINCT] col-1, col-2,col-3,.....,col-N
FROM table-1,....., table-N
[WHERE condition]
[ORDER BY column_1 [ASC|DESC][,column_2 [ASC|DESC]...]];
```

图 2-1 显示 DISTINCT 选项及缺省值 ALL 的 SELECT 语句

例 2.6

在 S_DEPT 表中，有多少个不同的部门？

由于我们不需要重复的部门名称，故有必要用 SELECT 子句的 DISTINCT 选项。为获得这个信息的查询以及它所产生的结果如下：

```
SELECT DISTINCT name
FROM s_dept;
NAME
--------------------
Administration
Finance
Operations
Sales
4 rows selected.
```

可观察到查询结果中没有重复行。

读者需注意的是：当投影作用于一列以上的关系时，行的惟一性由它所有的列值来决定。下例将对此进行说明。

例 2.7

显示 World_Cities 表中所有不同国家的国名及其所在的洲名。要求国名不能重复。

```
SELECT DISTINCT country, continent
FROM World_Cities;
COUNTRY                        CONTINENT
------------------------------ -------------
ARGENTINA                      SOUTH AMERICA
AUSTRALIA                      AUSTRALIA
BRAZIL                         SOUTH AMERICA
CANADA                         NORTH AMERICA
CHILE                          SOUTH AMERICA
CHINA                          ASIA
COLOMBIA                       SOUTH AMERICA
ECUADOR                        SOUTH AMERICA
EGYPT                          AFRICA
ENGLAND                        EUROPE
FRANCE                         EUROPE
GREECE                         EUROPE
INDIA                          ASIA
ITALY                          EUROPE
JAPAN                          ASIA
LIBYA                          AFRICA
PERU                           SOUTH AMERICA
POLAND                         EUROPE
RUSSIA                         EUROPE
SPAIN                          EUROPE
UNITED STATES                  NORTH AMERICA
VENEZUELA                      SOUTH AMERICA
22 rows selected.
```

注意,在所得到的表中有22个不同的行。正如前面讲到的,国名及洲名组合起来决定了行的惟一性,但并不要求任一列的值是惟一的。在CONTINENT列下,EUROPE重复了多次,但在所得表中,并无重复行。通过这个查询,可以看到DISTINCT选项的明显效果。如果该查询不用这个选项,所得到的表中就会有32行。必须清楚的一点是:如果没有DISTINCT选项,该查询不执行在World_Cities表上对属性country及continent的投影。

2.4 连接运算符的执行

连接(join)运算允许我们将来自两个或多个表的数据形成一个单一表。参与连接的表通过WHERE子句限定的条件而产生的共同属性连接起来。这些共同属性通常称为码-外码联系。连接的分类有以下方式:按所包含的条件的类型(等值还是非等值);按所包含的表的数量(只有一个表还是有多个表);按被检索的行的类型(在其他表中有无直接匹配)。在本章中,我们只考虑表2-2中指出的连接类型。

表2-2 部分连接类型列表

类型	说明
等值连接	两表通过共同的等值列而连接
自连接	一个表与其本身的连接
外连接	两个表之间的连接,决定一个表的所有行在另一表中没有匹配的元组

2.4.1 等值连接

等值连接是两表通过共同的等值而进行的连接。图2-2给出了执行这种连接类型的SELECT命令语法。该命令可轻易地将两个以上的表连接起来

```
SELECT table 1.col1,...,table-1.colN,...,table-2.col1,...,table-2.colN
FROM table 1, table-2
WHERE table-1.common column = table 2.common-column;
```

图2-2 连接两个表的SELECT语句的常用格式

注意,在SELECT子句中,我们在每一个列名的前面加上了它的表名。如果是在一个以上的表中出现同样的列名就必须这样做。在列名前加上表名的列称为被限定列。如果在两个表中没有相同的列名,则没有必要限定列。但是为了提高系统性能,还是对列加以限定为好。

出现在SELECT子句中的列名必须存在于FROM子句中提到的诸表之中,目前这一点是显而易见的。

例2.8

选择S_DEPT表中所有雇员的姓氏、名字及所在的部门名称。

要回答这个问题，我们需要将来自两个表(即 S_DEPT 表和 S_EMP 表)的数据放在一个 Join 表中。可从 S_EMP 表获得属性 last name 及属性 first name，而属性 department name 可从 S_DEPT 表获得。共同属性为 dept_id(S_EMP 表的属性)和 id(S_DEPT 表的属性)，正是通过这些共同属性，我们可将表连接并将所获得的信息放在一个单一的表中。定义该等值连接的 WHERE 子句的条件可表示为：

```
s_emp.dept_id = s_dept.id
```

注意，在本例中，属性 dept_id 是 S_EMP 表的外码，它是参照表 S_DEPT 的主码 ID。执行这个等值连接的 SELECT 语句如下：

```
SELECT s_emp.last_name, s_emp.first_name,s_dept.name AS
       "Department"
FROM s_emp,s_dept
WHERE s_emp.dept_id = s_dept.id;
```

可以看到，在这个查询中，我们限定了所有在 SELECT 及 WHERE 子句中提到的属性，即使这两个表并无共同的列名也是如此。由于这些属性名在任一表中均未重复，我们也可以用下述方式写出这个查询，该方式不会混淆 RDBMS[①]：

```
SELECT last_name, first_name, name AS "Department"
FROM s_emp,s_dept
WHERE s_emp.dept_id = s_dept.id;
```

该查询结果如下：

```
LAST_NAME            FIRST_NAME           DEPARTMENT
-------------------- -------------------- ---------------
Martin               Carmen               Administration
Smith                Doris                Operations
Norton               Michael              Sales
Quentin              Mark                 Finance
.
.
.
25 rows selected.
```

2.4.2　使用表别名简化查询

有时，我们想简化与限定属性有关的查询的书写形式。限定列名不仅费时，而且容易出错，特别是当连接的诸表具有较长名称时更是如此。为了简化此任务，我们可以用表别名代替

① 从这种意义上说，限定列是为了避免 RDBMS 产生任何含糊

表名。一个表别名是我们在 SELECT 命令中赋予表的一个暂时名。表别名在 SELECT 语句的 FROM 子句表名的后面。写表别名时，不需要专门的标点符号。在执行 SELECT 语句以后，该别名就不再存在。通过选择一个短的表别名，可以减少写一个查询时敲入的字符数。需要记住的一点是：一旦在 FROM 子句中给表赋予了一个别名，则该别名的使用必须贯穿整个 SELECT 命令。下例对表别名的使用进行了说明。

例 2.9

使用表别名，显示运动用品数据库中所有雇员的姓氏、名字及其所在的部门名。

该查询相当于例 2.8 的查询。为了便于比较，我们再演示一遍例 2.8 的查询。

```
SELECT s_emp.last_name, s_emp.first_name,s_dept.name
FROM s_emp,s_dept
WHERE s_emp.dept_id = s_dept.id;
```

使用别名(以粗体字显示)的同样的查询可以表示如下：

```
SELECT A.last_name, A.first_name, B.name
FROM s_emp A ,s_dept B
WHERE A.dept_id = B.id;
```

在 FROM 子句中，给表一个别名。表 S_EMP 的别名为 A，而表 S_DEPT 的别名为 B

注意，在 FROM 子句中，每个表名后都跟着一个别名，表 S_EMP 的别名为 A，而表 S_DEPT的别名为 B。因此在整个 SELECT 命令中，我们均用 A 代替 S_EMP，而用 B 代替 S_DEPT

表别名可长达 30 个字符，但是使用这么长的别名与简化查询的目的相悖。笔者认为只要使用表别名，总是"越短越好"！

2.4.3 使用文字及并置运算符修改查询结果

列别名的使用，允许我们对一个查询结果的标题进行控制，但对行如何显示则无能为力。使用并置运算符才能做到这一点。并置运算符以两个垂直线 || 表示。并置运算符将列名或字符串或者列名和字符串二者放在一起，形成一个字符表达式。一个字符串是在一对单引号内的一个字符序列。为了强化查询的显示，我们将并置运算符与一个或多个文字字符联合起来使用。文字字符是指在 SELECT 语句中，除了列名或别名以外的任意字符串、表达式或者数下面的例子说明了使用并置运算符及文字字符来修改简单的查询结果。在第 4 章我们将进一步讲解并置运算符的使用。

例 2.10

显示 S_EMP 表中所有雇员的姓氏、名字及职务

下面的 SELECT 命令告诉我们如何将列名及文字字符并置，形成一个字符表达式。注

意，字符表达式被当作一个可以用别名重新命名的单一列。使用空格将字符串与列值分开，以提高查询结果的可读性。该查询及部分结果如下所示：

```
SELECT first_name || ' ' || last_name || ' ' || 'is a ' ||
title || ' of the company' AS "Employees and their titles"
FROM s_emp;

Employees and their titles
--------------------------------------------------
Carmen Martin is a President of the company
Doris Smith is a VP, Operations of the company
Michael Norton is a VP, Sales of the company
.
.
25 rows selected.
```

2.4.4　自连接

自连接是一个表与其自身的连接。如果我们给同一个表两个不同的别名，就可很容易地使用这种类型的连接。这样，我们就将同一个表看作两个不同的表。下例对此进行了讲解：

例 2.11

显示运动用品数据库中每一个雇员的名字及他或她的管理者的名字。

为回答这个查询，我们需要将表与它自身连接。如前所述，如果我们给同一个表两个不同的别名，就可使用这种类型的连接。在本例中，我们指定 S_EMP 表的别名为 E(对雇员)及 M(对管理者)。检索此信息的查询及结果如下所示：

```
SELECT E.last_name || ' works for ' || M.last_name || ' ' ||
E.first_name AS "Employees and their managers"
FROM s_emp M, s_emp E
WHERE E.id = M.manager_id;

Managers and their employees
-------------------------------------
Smith works for Martin Carmen
Norton works for Martin Carmen
Quentin works for Martin Carmen
Roper works for Martin Carmen
.
.
24 rows selected.
```

2.4.5 外连接

假设有两个表的连接，有时我们想知道不满足其中一个表指定条件的另一个表的行。例如，我们想知道还没有管理者的雇员。能够回答这类问题的运算符是外连接运算符。该运算符的语法如图 2-3a 及图 2-3b 所示。

```
SELECT table1.col1,...table1.colN, table1.col1,...table2.colN
FROM table1, table2
WHERE table1.column-name(+) = table2.column-name;
```

图 2-3a 用于计算在表 1 中所有与表 2 不匹配的行的外连接

```
SELECT table1.col1, ...table1,colN, table1.col1,....table2.colN
FROM table1, table2
WHERE table1.column-name = table2.column-name(+);
```

图 2-3b 用于计算在表 2 中所有与表 1 不匹配的行的外连接

外连接运算符由(+)号表示，该运算符用在 WHERE 子句里，(+)接在与另一表中有不匹配行的表的列名后面。按照这个运算符的语法，(+)可置于 WHERE 条件的任一列名后，但不可在所有列名后都用(+)号。下例讲解了该运算符是如何起作用的。

例 2.12

列出无管理者的所有雇员。

回答这个问题的查询及部分结果如下：

```
SELECT E.last_name || ' ' || 'works for' ||' '||M.last_name ||' '||
E.first_name AS "Managers and their employees"
FROM s_emp M, s_emp E
WHERE E.id (+)= M.manager_id;

Managers and their employees
-----------------------------
Martin works for <-------- 注意，Martin没有管理者。

Smith works for Martin Carmen
Norton works for Martin Carmen
Quentin works for Martin Carmen
.
.
.
25 rows selected.
```

可观察到，除了在 WHERE 子句的条件中出现了外连接运算符外，这个查询几乎与上例的查询一致。外连接运算符可产生一行或多行能与另一表的一行或多行连接的空行，结果中含有未连接前表中所没有的元组。在此例中，显示了 Martin Carmen 没有管理者的元组。

2.5　建立外码

在第 1 章中，我们讲解了 CREATE TABLE 命令，了解了如何定义一个简单的主码及如何说明 NOT NULL 或 UNIQUE 属性。但我们未讲解如何强化完整性约束。如 1.4 节所述，外码用于在两个关系的行间或同一关系的行间维护数据的一致性。外码可以在创建表时或者在表创建后进行定义。我们将对这两种情况进行讲解。

2.5.1　在创建表时定义外码

可以在列一级或者在表一级上定义一个表内的外码。如在第 1 章所讲，在列一级上，对表的任意属性只能定义一个约束。在列一级上定义一个外码的语法如下所示。读者必须记住，在任一表中，可能存在一个或多个属性被定义为 FK。在建立任何一个 FK 之前，用户必须确保参照母表的列或者是一个 PK，或者是已被定义为 UNIQUE。在列一级定义外部码的语法如下所示：

```
CREATE TABLE table-name
(
column-name-1      data type-1 [CONSTRAINT constraint-name]
                   REFERENCES Parent-Table(column-of-parent-
                   table),
 column-name-2     data type-2 [constraint],
                   .
                   .
 column-name-N     data type-N [constraint],
);
```

例 2.13

根据下列信息，建立 CUSTOMERS 表。定义 SalesRep 为 FK，且该 FK 参照 SalesAssociates 表(此处未给出)的属性 AssociateId。这里需要进行什么设定呢？

Customer ID	City	Credit	SalesRep	Last Order
0109	Miami	Good	9012	1/12/00
0245	Caracas	Good	9786	1/15/99
0345	London	Poor	9873	1/26/99

创建这个表的命令如下：

```
CREATE TABLE customer(
  CustomerId   Varchar2(5) NOT NULL,
  City         Varchar2(25),
  Credit       Varchar2(4),
  SalesRep     Varchar2(4) REFERENCES SalesAssociate
               (AssociateId),
  LastOrder    Date);
```

父表表名

父表中被参照的列

在加上这个外码约束之前，必须知道在 SalesAssociate 表中，属性 AssociateId 已经被定义为 PK 或者 UNIQUE。

2.5.2 在一个已存在的表中定义外码

要在一个已存在的表中定义外码约束，我们需要使用 ALTER TABLE 命令，为了用 ALTER TABLE 命令定义一个单一的 FK 约束，我们将使用该命令的下列变体：

```
ALTER TABLE table-name
ADD [CONSTRAINT constraint-name] FOREIGN KEY(column-name)
REFERENCES parent-table (parent-table column);
```

例 2.14

使用上例的 CUSTOMERS 表，将一个 FK 加至此表中，使属性 Credit 参照 CreditType 表的属性 Credit_Standing。假设以前已经定义了 CreditType 表，但并未在这里展示。需要什么样的设定，才能确保可以在 CUSTOMERS 表中定义外码呢？

为将一个约束加到已存在的表中，我们需要使用 ALTER TABLE 命令，欲加上所需要的约束，ALTER TABLE 命令如下所示：

```
ALTER TABLE customer
ADD CONSTRAINT CreditType_Credit_FK FOREIGN KEY (CREDIT)
REFERENCES CreditType (CreditStanding);
```

为了确保 FK 的建立，属性 Credit_Standing 必须是 CreditType 表中的 PK 或 UNIQUE 属性。

2.6 在一个已存在的表中定义主码

对一个已建立的表，可以使用 ALTER TABLE 命令为其定义一个单一属性或属性组合的主码。把一个 PK 约束加到表中的 ALTER TABLE 命令的语法如下：

```
ALTER TABLE ADD [CONSTRAINT constraint-name]
PRIMARY KEY (column-1 [,column-2 [,column-3]]...] );
```

注意，如果主码是一个以上属性的组合码，在圆括号里要提到这些属性，属性之间用逗号隔开。下面的例子将说明该命令的用法

例 2.15

假设 Baseball_World_Series_Result 表已建立，加入相应的约束，使属性 Year 成为该表中的单一属性主码。该表的结构如下：

Attribute	Data Type
Year	Data
American_League_Team	Varchar2(30)
National_League_Team	Varchar2(30)
Winner_Manager	Varchar2(25)
Losing_Manager	Varchar2(25)
Most_Valuable_Player	Varchar2(25)
Games_Played	Char(1)

使属性 Year 成为该表的 PK 的命令如下：

```
ALTER TABLE Baseball_World_Series_Result
ADD PRIMARY KEY (year);
```

例 2.16

表 Summer_Olympics_Games 的结构如下所示，定义该表的组合主码带有属性 Year 和属性 Country_Name。

Attribute	Data Type
Country_Name	Varchar2(25)
Year	Data
Gold	Number(3)
Silver	Number(3)
Bronze	Number(3)

为建立所需要的组合主码的 ALTER TABLE 命令如下所示：

```
ALTER TABLE Summer_Olympics_Games ADD
PRIMARY KEY (Country_Name, Year);
```

由于该表的码是两个属性的组合码，因此这两个属性均需在 PRIMARY KEY 子句中提到。

2.7 使用 CHECK 约束限制属性列的输入值

在前面所有的例子中，我们均假设填于表中的数据都具有合适的大小、范围及数据类型。但是情况并非总是如此，因为用户在输入数据的过程中会出错。例如：当从一个客户处拿到一份定单时，数据录入操作员可能会输入一份超过该客户被允许的信用数量的订单。在这种情况下，最好能在输入数据的同时检查出这些错误。为了避免将不正确的值输入表中，我们可对表的属性加上一个 CHECK 约束。尽管可以在把表填充完之后加上 CHECK 约束，但最好还是在输入数据之前定义这些约束类型。这样，我们就能尽可能早地察觉到输入的错误。下例讲述了 CHECK 约束的使用。

例 2.17

假设我们正为一些客户建立 Mailing_List 表，这些客户居住地的邮编分别为：22801、22802、22803 及 22804。为了避免输入错误的邮编，我们可在 Mailing_List 表中建立一个 CHECK 约束，如下所示：

```
CREATE TABLE Mailing_list(
  First_name   Varchar2(25),
  Last_name    Varchar2(25),
  Address      Varchar2(25),
  City         Varchar2(25),
  Zip_code Varchar2(5) CHECK( zip_code IN ('22801',
 '22802', '22803',  22804')));
```

注意，在此例中，我们使用了一个 CHECK 约束，限制可插入到 zip_code 列的值。逻辑运算符 IN 允许我们明确表达可被插入该列的一些值。在此例中，我们将允许的输入值置于单引号内，这是因为属性 zip_code 是一个字符型属性。有关逻辑运算符 IN 的其他信息，请参见第 3 章 3.3.2 节。

例 2.18

假设我们正在建立一个 Department 表，其中，部门数大于 10，小于 50。写出 CHECK 约束，以确保插入表中的每个部门数都在这个范围之内。

在此例中，相应的 CREATE TABLE 命令如下：

```
CREATE TABLE department(
 Name VARCHAR2(15) CONSTRAINT department_name_nn NOT NULL,
 Location  VARCHAR2(20),
 DeptNum   NUMBER CHECK (DeptNum BETWEEN 10 and 50));
```

在此例中，我们运用了逻辑运算符 **BETWEEN** 给 DeptNum 属性设立允许的数值范围。有关逻辑运算符 BETWEEN 的其他信息，请参见第 3 章 3.3.1 节。

2.8　对已存在的表添加属性列

有时,需要在一个已存在的表中添加一个新的属性列。为做到这一点,我们需要使用 ALTER TABLE 命令的 ADD 选项。该命令的语法如下:

```
ALTER TABLE table-name
ADD new-column data-type [CONSTRAINT] [constraint-type];
```

当向表中加入一个新列时,读者需记住,新列的每一行最初的值均为 NULL。只有当一个表还没有行的时候,我们才可以对添加的列用 NOT NULL 约束。

例 2.19

在例 2.18 的 Department 表中,加入一个名为 Manager_LastName 的新列,假设该列可长达 25 个字符。

添加该新列的 ALTER TABLE 命令如下:

```
ALTER TABLE department
ADD Manager_LastName VARCHAR2(25);
```

2.9　对已存在的表修改属性列

如果需要对一个已创建的表修改列的定义,可以使用 ALTER TABLE 命令的 MODIFY 选项。该命令的基本语法如下:

```
ALTER TABLE table-name
MODIFY column-name [CONSTRAINT constraint-name] [constraint-type];
```

读者必须记住,在使用这个命令时有一些限制,我们只能改变下面的列特性:数据类型、大小、缺省值及 NOT NULL 列约束。MODIFY 子句只需要列名及修改的部分,而不需要整个列的定义。现将使用该命令的限制总结如下:

- 如果该列的所有行全为 NULL 值时,可将 CHAR 改为 VARCHAR2 或将 VARCHAR2 改为 CHAR。
- 如果表中所有行包含 NULL 值时,可以改变数据类型及减小列的大小。
- 如果列不包含 NULL 值时,用 MODIFY 选项可加入表中的惟一约束是 NOT NULL 约束。
- 增加字符数的长度或者数字列的精度总是可行的。

例 2.20

在 Department 表中,增加 name 列大小,以便可添入新建立的部门"Operations Research

and Marketing Analysis"

相应的 ALTER TABLE 命令如下：

```
ALTER TABLE department MODIFY name VARCHAR(45);
```

2.10 从表中删除约束

要从表中删除一个已有的约束，我们可使用 ALTER TABLE 命令中的 DROP 选项。该命令的变体用于删除未命名的或命名的约束，语法如下：

从已知表中删除 PK，使用下列命令：

```
ALTER TABLE table-name DROP PRIMARY KEY [CASCADE];
```

删除一个未命名的 UNIQUE 约束，使用下列命令：

```
ALTER TABLE table-name DROP
UNIQUE(column[,column[,column...]]..) [CASCADE];
```

删除一个命名的 UNIQUE 约束，使用下列命令：

```
ALTER TABLE table-name DROP CONSTRAINT constraint-name [CASCADE];
```

在所有上述命令中，CASCADE 子句删除建立在已删除的完整性约束上的任一约束。如果不用这个子句，在不先删除外部码的情况下，我们不能删除作为参照完整性约束一部分的 UNIQUE 或 PK。此外，还要注意在这个命令中，没有专门的选项用于删除 CHECK 或者 NOT NULL 约束。但是，有一个命令可删除任意命名的约束。如果我们给 CHECK 或 NOT NULL 约束命名的话，就可使用 ALTER TABLE 命令的变体删除它们。

例 2.21

在例 2.18 中，删除 Department 表中 name 属性的 NOT NULL 约束。

相应的 ALTER TABLE 命令如下：

```
ALTER TABLE department DROP CONSTRAINT department_name_nn;
```

注意，在此例中，我们删除的是约束名，而不是某些具体的约束。这就是使用约束名的优点之一，因为我们只需要记住 ALTER TABLE 命令适用于各种情况的一个特例。

问题与答案

注意：在回答下列问题之前，请运行脚本 SG.SQL，以刷新运动用品数据库的内容。

2.1 写出显示客户 204 发出的全部订单的查询。给出订单 ID、每份订单的合计以及订单发

出的时间。这个查询执行什么类型的关系运算符？请使用 s_ord 表。

获得所需信息的查询及它产生的结果如下。所用的关系运算符为投影运算符。

```
SELECT id, total, date_orderec
FROM s_ord
WHERE customer_id = '204';

ID      TOTAL DATE_ORDE
--- --------- ---------
100    601100 31-AUG-92
111      2770 09-SEP-92
2 rows selected.
```

2.2　重新写出上述查询，要求标题显示为：Order Id、Total Ordered 及 Order Date。确保订货合计以格式 ddd,ddd.00 显示(此处 d 代表 1 个数字)且标题分两行书写。

由于属性 id 定义的字符数对显示标题 Order Id 太少，因此我们需要使用 SQL * Plus COLUMN 命令。同样，我们需要使用所要求的数据格式来显示合计。所需要的 COLUMN 命令、SELECT 查询及其结果如下所示：

```
COLUMN id HEADING "Ordered | Id" FORMAT A7
COLUMN total HEADING "Total | Ordered' FORMAT 999,999.00
COLUMN date_ordered HEADING "Order|Date"
SELECT id, total, date_ordered
FROM s_ord
 WHERE customer_id = '204';
```

← 新标题

```
Ordered     Total   Order
Id        Ordered   Date
------- ---------- ---------
100     601,100.00 31-AUG-92
111       2,770.00 09-SEP-92
2 rows selected.
```

2.3　显示 S_ORD 表中有多少个不同的客户已提交了定单。重命名列 Customers who have placed an order，并将标题分两行书写。要回答这个查询，应执行什么类型的关系运算符？

在发出这个查询之前，有必要对用户列进行格式化，以改变其标题。下述 SQL * Plus 命令允许我们按要求显示列标题。

```
COLUMN customer_id HEADING "Customers who have |placed an order" FORMAT A20
```

回答这个查询所用的关系运算符为投影运算符。在 SQL 中，执行这个运算符及所得的结果表如下所示。

```
SELECT DISTINCT customer_id
FROM s_ord;
Customers who have
placed an order
```

```
--------------------
201
202
203
204
205
206
208
209
210
211
212
213
214
13 rows selected.
```

2.4 在不写出表的列名的情况下，给出表 S_REGION 中的所有信息。

这个查询及其结果如下所示：

```
SELECT *
FROM s_region;
Id     NAME
------- --------------------
1      North America
2      South America
3      Africa / Middle East
4      Asia
5      Europe
6      Central America /Caribbean

6 rows selected.
```

2.5 显示 S_EMP 表中所有销售代理的姓氏、工资及佣金百分率。使用标题 Sales Representative 代替 Last Name。该查询必须使用 SQL * Plus COLUMN 命令吗?

获得这些信息的查询以及所产生的结果如下所示。该查询并不是必须要使用 SQL * Plus COLUMN 命令，因为属性 last_name 的宽度足够大，可显示列别名 Sales Representative。

```
SELECT last_name AS 'Sales Representative', salary,
commission_pct
FROM S_emp
WHERE title = 'Sales Representative';

Sales
Representative          SALARY COMMISSION_PCT
-------------------- --------- --------------
Henderson                 1400             10
Gilson                    1490           12.5
Sanders                   1515             10
Dameron                   1450           17.5
4 rows selected.
```

2.6　显示运动用品数据库中所有部门的名称及它们所在的区域。获得这些信息，需要用到什么类型的关系运算符？

欲检索这些信息，我们需要将表 S_REGION 和表 S_DEPT 连接起来，它们的共同属性为 region_id(在 S_DEPT 表)及 id(在 S_REGION 表)。注意使用表的别名来简化查询。本查询及所产生的结果如下：

```
SELECT D.name AS "Department", R.name AS "Region"
FROM s_dept D, s_region R
WHERE D.region_id = R.id;
Department           Region
-------------------- --------------------
Finance              North America
Sales                North America
Sales                South America
Sales                Africa / Middle East
Sales                Asia
Sales                Europe
Operations           North America
Operations           South America
Operations           Africa / Middle East
Operations           Asia
Operations           Europe
Administration       North America
12 rows selected.
```

2.7　显示所有销售代表及他们的客户的姓氏、名字。

获得这些信息的查询如下所示。注意，并置运算符、字符串及空格的使用会改变结果表的外观。同时观察表别名的使用对查询的简化。

```
SELECT C.name || ' is the customer of ' || E.first_name ||
' ' || E.last_name AS "Sales Reps and their customers"
FROM s_emp E, s_customer C
WHERE C.sales_rep_id = E.id
ORDER BY E.last_name;
Sales Reps and their customers
--------------------------------------------------
Toms Sporting Goods is the customer of Andre Dameron
Athletic Attire is the customer of Andre Dameron
Athletics One is the customer of Andre Dameron
Athletics Two is the customer of Andre Dameron
Shoes for Sports is the customer of Andre Dameron
Sports,Inc is the customer of Sam Gilson
.
.
24 rows selected.
```

2.8 显示目前还没有指定销售代表的客户。

外连接可满足这个查询的要求,如下所示:

```
SELECT C.name || ' is the customer of ' || E.first_name ||
' ' || E.last_name
AS "Sales Reps and their customers"
FROM s_emp E, s_customer C
WHERE C.sales_rep_id = E.id(+)
ORDER BY E.last_name;
Sales Reps and their customers
--------------------------------------------------
Toms Sporting Goods is the customer of Andre Dameron
Athletic Attire is the customer of Andre Dameron
.
.
Hamada Sport is the customer of Jason Sanders
Muench Sports is the customer of Jason Sanders
Tall Rock Sports is the customer of          <-- 没有销售代表的客户
.
.
25 rows selected.
```

2.9 显示 S_EMP 表中所有雇员的姓氏及工资。确保工资以 999,999.00 的格式显示,此处每个 9 均代表一个数字。结果按工资数的升序排列。

为了使结果以所要求的格式显示,有必要使用 SQL * Plus COLUMN 命令。查询及部分结果如下所示。注意,在 WHERE 子句中,没有必要使用保留关键字 ASC,因为它是缺省值。

```
COLUMN salary FORMAT 999,999.00
SELECT last_name AS "Last Name", first_name AS "First
Name", salary AS "Salary"
FROM s_emp
ORDER BY salary;
Last Name            First Name           Salary
-------------------- -------------------- -------
Patterson            Donald                795.00
Pearl                Roger                 795.00
Gantos               Eddie                 800.00
Chester              Eddie                 800.00
Bell                 Alexander             850.00
.
.
25 rows selected.
```

2.10 按以下的描述建立表 Lakes_of_The_World,在必要的地方使用命名约束。

属性	数据类型	约束
Name	最长 20 个字符	主码
Continent	最长 15 个字符	取以下的值之一：Africa、Asia、Australia、Europe、Central America、North America、South America
Area	最长 6 位数字	值在 1799 及 143 244 之间
Length	最长 3 位数字	值在 67 及 760 之间
Elevation	最长 4 位数字	值在 -92 及12 500之间

创建此表的 CREATE TABLE 命令如下：

```
CREATE TABLE Lakes_Of_The_World (
Name VARCHAR2(20) PRIMARY KEY,
Continent VARCHAR2(15) CONSTRAINT Lotw_Continent_ck
CHECK( Continent IN ( 'Africa', 'Asia', 'Australia' ,
                      'Europe', 'Central America',
                      'North America', 'South America')),
Area       Number(6) CONSTRAINT Lotw_Area_ck
                CHECK ( Area between 1799 AND 143 244),
Length     Number(3) CONSTRAINT Lotw_Length_ck
                CHECK(Length between 67 AND 760),
Elevation Number(4) CONSTRAINT Lotw_Elevation_ck
                CHECK( Elevation between -92 AND
                       123500));
```

2.11　将新列(列名为 Performance)加至 S_EMP 表。该列的允许值为'Satisfactory'、'Excellent'和'Unsatisfactory'。证实该列被正确地加入了。

欲对一个已存在的表添加一个新列，我们需要使用一个 ALTER TABLE 命令。此时，该命令如下所示：

```
ALTER TABLE s_emp ADD performance VARCHAR2(15) CONSTRAINT
s_emp_performance_ck CHECK (performance IN ('Satisfactory',
'Excellent', 'Unsatisfactory'));
```

2.12　将 S_CUSTOMER 表中的属性 address 的长度从 20 个字符增加到 25 个字符。

为了增加这个表中一个属性的大小，需要使用 ALTER TABLE 命令。此时，该命令格式如下所示　观察一下可以发现，与该列相关的其他约束依然保持不变。

```
ALTER TABLE S_customer
MODIFY address VARCHAR2(25);
```

2.13　在问题与答案 2.11 中，删除添加到 S_EMP 表中的 CHECK 约束。

从表中删除一个命名约束时，需要使用 ALTER TABLE 命令　此时，该命令如下所示：

```
ALTER TABLE S_emp DROP CONSTRAINT s_emp_performance_ck;
```

2.14　显示表 S_CUSTOMER 中仓库管理员的姓氏及名字。

为了检索这个信息,我们需要将 S_EMP 表及 S_CUSTOMER 表的共同属性做等值连接。共同属性是 manager_id(S_CUSTOMER 表)及 id(S_EMP 表)。等值连接及其产生的结果如下:

```
SELECT E.first_name || ' ' || E.last_name ||
                        ' is the manager of warehouse ' || W.id
                       AS "Warehouses and their managers"
FROM S_warehouse W, S_emp E
WHERE W.manager_id = E.id;
Warehouses and their managers
-----------------------------------------------------------
Molly Brown is the manager of warehouse 101
Marta Jackson is the manager of warehouse 10501
Roberta Hawkins is the manager of warehouse 201
Ben Burns is the manager of warehouse 301
Antoinette Catskill is the manager of warehouse 401
```

2.15 目前你已建立了几个表及它们各自的 FK 约束,但当你尝试填表时,却得到了参照性约束错误,为什么?

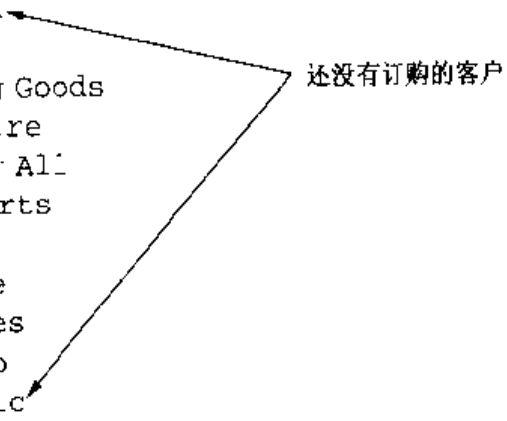
当表与表之间交叉引用时,可能发生此类错误。因为表 A 可能有一个参照表 B 的 FK,同时,表 B 有一个 FK 是参照表 A 的。为了避免此类错误,应该在诸表已填充后,再建立外部码。

2.16 写出显示所有客户及他们定单的查询。如果有客户还没有订购,确保也显示他们的名字。请使用以下标题:Customer Id、Name 及 Order Id。

欲获得这些信息,我们需要对 S_CUSTOMER 表及 S_ORD 表进行外连接。该连接及产生的结果如下所示。注意,SQL * Plus COLUMN 命令及列别名的联合使用可以得到所要求的标题。

```
COLUMN A HEADING 'Customer Id' FORMAT A15
COLUMN B HEADING 'Order Id' FORMAT A8
SELECT S_customer.id AS "A",
       S_customer.name, S_ord.id AS "B"
FROM s_customer , s_ord
WHERE S_customer.id = S_ord.customer_id(+);

Customer Id     NAME                 Order Id
--------------- -------------------- --------
201             One Sport            97
202             Deportivo Caracas    98
203             New Delhi Sports     99
204             Ladysport            100
204             Ladysport            111
205             Kim's Sporting Goods 101
206             Sportique            102
207             Tall Rock Sports     <-- 该客户还没有订购
```

```
208          Muench Sports          103
208          Muench Sports          104
209          Beisbol Si!            105
210          Futbol Sonora          106
210          Futbol Sonora          112
211          Helmut's Sports        107
212          Hamada Sport           108
213          Sports Emporium        109
214          Sports Retail          110
215          Sports Russia
301          Sports,Inc
302          Toms Sporting Goods
303          Athletic Attire
304          Athletics For All
305          Shoes for Sports
306          BJ Athletics
403          Athletics One
404          Great Athletes
405          Athletics Two
406          Athletes Attic
28 rows selected.
```



2.17　用下面给出的属性写出 Class 表。该表的主码由属性 Course_Code 和 Class_Section 组合而成

属性	数据类型	约束
Course_code	最多 5 个字符	需要,非空值
Class_Section	最多 4 个字符	需要,非空值
Classroom	最多 5 个字符	
Instructor	最多 25 个字符	
Max_Enrollment	2 位数字	10 到 30 之间

由于主码是一个组合码,因此有必要在表一级定义 PK。相应的 CREATE TABLE 命令如下所示:

```
CREATE TABLE Class
( Course_Code      VARCHAR2(5),
  Class_Section    VARCHAR2(4),
  Classroom        VARCHAR(5),
  Instructor       VARCHAR2(25),
  Max_Enrollment NUMBER(2) CONSTRAINT Class_Max_Enroll_ck
                   CHECK (Max_Enrollment BETWEEN 10 AND 30),
  CONSTRAINT Class_Code_Section_pk PRIMARY KEY(Course_Code,
                                               Class_Section));
```

2.18 在上例的 Class 表中加入一个新列。该列名为 Teaching Assistant;该列长度可多达 15 个字符。

允许我们加入这个新列的命令是 ALTER TABLE 命令。此时,该命令的语法如下:

```
ALTER TABLE class ADD Teaching_Assistant VARCHAR2(15);
```

2.19 写出能将 Teaching_Assistant 列的长度增加到 VARCHAR2(25) 的相应命令。

允许我们增加列的长度的命令是 ALTER TABLE 命令。此时,该命令的语法如下:

```
ALTER TABLE class MODIFY Teaching_Assistant VARCHAR2(25);
```

2.20 在 Class 表中,删除与属性 Max_Enrollment 相关的约束。

欲删除一个命名约束,必须使用 ALTER TABLE 命令。该命令的语法如下:

```
ALTER TABLE class DROP CONSTRAINT Class_Max_Enroll_ck;
```

补 充 题

注意:在回答下列问题前,请刷新运动用品数据库。

2.21 使用外查询,找出 Sporting Goods 公司没有客户的雇员的名字。

2.22 显示 S_EMP 表中所有雇员的姓氏、名字及工资。确保结果中的每一行用以下格式显示:

The salary in U.S. dollars of Donald Patterson is 795.

The salary in U.S. dollars of Roger Pearl is 795.

2.23 按照下列要求建立表 RIVERS。

属性	数据类型	约束
Name	最长 20 个字符	惟一且非空
Length	最长 4 位数字	值在 100 至 4160 之间
Outflow	最长 20 个字符	需要的

2.24 将一名为 MaxDepth 的新列加至 Rivers 表中。确保该列值的范围在 100 至 250 英尺之间。

2.25 运行 World_Cities 脚本,以字母顺序显示所有南美城市的名称。

2.26 从 S_item 表中删除约束 S_ITEM_PRODUCT_ID_FK。

2.27 显示在 Sporting Goods 公司工作的全体雇员的姓氏、名字及部门名称。用以下列格式显示结果:

Employee Jason Sanders works for department 33.

2.28 将上一题查询结果的标题改为 Employees and their departments

2.29 修改上一题的查询,将结果用以下模式显示:

Employee Jason Sanders works for the Sales Department.

2.30　显示仓库的 ID 及它们所在的地区。

2.31　显示产品名称、产品 ID 及所有产品的定货数。根据产品名称将结果排序。

2.32　显示建议批发价均高于 100 美元的所有产品名称及建议批发价。确保价格以下列模式显示：

World Cup Net　　$123.00

Bunny Boot　　$150.00

分别使用标题 Product Name 和 Suggested Wholesale Price，后者分成两列书写。将结果按照建议批发价的升值排序。

2.33　显示不同仓库的 ID 及它们所在的国家。用下列格式显示结果：

Warehouse 100 is located in Seattle, USA.

2.34　删除表 S_CUSTOMER 的约束 s_customer_id_pk，会出现什么情况？为什么？

2.35　为了删除上一问题中的约束，应该怎么做？

补充题答案

2.21

```
SELECT C.name || ' is the customer of ' || E.first_name ||
' ' || E.last_name
AS "Sales Reps and their customers"
FROM s_emp E, s_customer C
WHERE C.sales_rep_id(+) = E.id
ORDER BY E.last_name;
```

2.22

```
SELECT 'The salary in U.S. dollars of ' || first_name ||
' ' || last_name || ' is ' ||salary AS
"Employee Salaries"
FROM s_emp
ORDER BY salary;
```

2.23

```
CREATE TABLE Rivers (
Name    VARCHAR2(20) PRIMARY KEY,
Length  NUMBER(4) CONSTRAINT Rivers_Length_ck
                  CHECK (Length BETWEEN 100 AND 4160),
Outflow VARCHAR2(20) CONSTRAINT Rivers_Outflow_nn
                     NOT NULL);
```

2.24

```
ALTER TABLE rivers ADD MaxDepth NUMBER(3) CONSTRAINT
rivers_maxdept_ck CHECK (MaxDepth BETWEEN 100 and 250);
```

2.25

```
SELECT city
FROM world_cities
WHERE continent = 'SOUTH AMERICA'
ORDER BY CITY;
```

2.26

```
ALTER TABLE s_item DROP CONSTRAINT S_ITEM_PRODUCT_ID_FK;
```

2.27

```
SELECT 'Employee '||first_name||' '||last_name||' '
       ||'works for department ' || dept_id
FROM s_emp;
```

2.28

```
CLEAR COLUMNS
COLUMN A HEADING 'Employees and their department'
FORMAT A80
SELECT 'Employee '||first_name||' '||last_name||' '
       ||'works for department ' || dept_id AS "A"
FROM s_emp;
```

2.29

```
COLUMN A HEADING 'Employees and their department'
FORMAT A70
SELECT 'Employee '||E.first_name||' '||E.last_name||'
'||'works for the '|| D.name || ' Department' AS "A"
FROM s_emp E, s_dept D
WHERE E.dept_id = D.id;
```

2.30

```
SELECT W.id || ' is located in region '|| R.id AS
"Warehouses and their regions"
FROM s_warehouse W, s_regior R
WHERE W.region_id = R.id;
```

2.31

```
SELECT s_product.name, s_product.id, S_item.quantity
"ORDERED" FROM s_product, s_item
WHERE s_product.id = s_item.product_id
ORDER by s_product.name;
```

2.32

```
COLUMN Price HEADING "Suggested|Wholesale Price"
FORMAT $999,999.00
SELECT P.name AS "Product Name", P.suggested_whlsl_price AS
"Price"
FROM s_product P
WHERE P.suggested_whlsl_price >100
ORDER BY P.suggested_whlsl_price;
```

2.33

```
COLUMN A HEADING 'Warehouses and their locations'
FORMAT A60
SELECT 'Warehouse ' || W.id || ' is located in ' || W.city
|| ', ' || W.country AS "A"
FROM s_warehouse W;
```

2.34

该约束不能被删除,因为列 ID 被数据库中其他表的外码所参照。

2.35

```
ALTER TABLE s_customer DROP CONSTRAINT s_customer_id_pk CASCADE;
```

第3章　布尔运算符和字符匹配

3.1　布尔运算符及在复合子句中的应用

SELECT语句的WHERE子句允许按照某个指定列的值检索元组。然而有时你可能想使用一个以上的列的值来检索元组。例如：在工资单表中，你想了解在公司工作10年以上、以小时计生产率的所有人的姓名；在计费表中，你想知道居住在某个邮区付费晚了的人名。布尔运算用于在SQL语句的WHERE子句中建立复合条件。复合条件是指含有两个或多个布尔运算符的条件。这些运算符为AND、OR和NOT。表3-1给出了下一节中将会用到的客户表S_CUSTOMER的子集。

表3-1　客户表

ID	NAME	ZIP_CODE	CREDIT_RA	SAL
301	Sports, Inc	22809	EXCELLENT	12
302	Toms Sporting Goods	22809	POOR	14
303	Athletic Attire	22808	GOOD	14
304	Athletics For All	22808	EXCELLENT	12
305	Shoes for Sports	22809	EXCELLENT	14
306	RJ Athletics	22810	POOR	12
403	Athletics One	17601	GOOD	14
404	Great Athletes	17602	EXCELLENT	12
405	Athletics Two	17602	EXCELLENT	14
406	Athletes Attic	17601	POOR	12

3.1.1　AND逻辑运算符

AND逻辑运算符用于连接SELECT语句的WHERE子句中的两个布尔条件。当执行这个SQL语句时，只能检索到同时满足这两个条件的行。用AND连接两个布尔条件的语法如下：

```
SELECT column_list
FROM tablename
WHERE condition_1 AND condition_2;
```

例3.1

写出一个SQL查询，列出S_CUSTOMER表中具有EXCELLENT信誉等级、邮政编码为22809的所有客户的名称。

```
SELECT name
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
AND zip_code = '22809';
```

所得结果表中列出了 Sports、Inc 和 Shoes for Sports 两个商店。在 S _ CUSTOMER 表中，有三个商店具有 EXCELLENT 的信誉等级，但只有两个所在地区的邮政编码为 22809。

例 3.2

写出一个 SQL 查询，列出查询 S _ CUSTOMER 表中具有 EXCELLENT 信誉等级且与 Sales rep id 为 12 的销售代表相关的所有客户的名称。

```
SELECT name
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
AND sales_rep_id = '12';
```

所得结果表列出三个不同的商店，分别名为：Great Athletes、Athletics for All 和 Sports, Inc，因为它们同时满足两个条件的复合布尔语句。

AND 可连接任意数量的子句。对结果表中包含的每一行来说，所有的条件都必须为真。AND 条件有互换性，也就是说条件 WHERE condition _ 1 AND condition _ 2 与条件 WHERE condition _ 2 AND condition _ 1 是相同的。

例 3.3

写出一个 SQL 查询，列出 S _ CUSTOMER 表中具有 EXCELLENT 信誉等级、邮政编码为 22809 且与 Sales rep id 为 12 的销售代表相关的所有客户名。

```
SELECT name
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
AND zip_code = '22809'
AND sales_rep_id = '12';
```

此次只有 Sports，Inc 满足所有三个条件。

3.1.2 OR 逻辑运算符

OR 逻辑运算符也是用于连接 SELECT 语句的 WHERE 子句中的两个布尔条件。当执行这个 SQL 语句时，可检索到满足其中一个条件或两个条件都满足的行。用 OR 连接两个布尔条件的语法如下：

```
SELECT column_list
FROM table_name
WHERE condition_1 OR condition_2;
```

与 AND 条件一样，OR 条件也是可互换的。

例 3.4

写出一个 SQL 查询，列出 S_CUSTOMER 表中具有 EXCELLENT 或 GOOD 信誉等级的所有客户名。

```
SELECT name
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
OR credit_rating = 'GOOD';
```

有五个商店有 EXCELLENT 信誉等级，有两个商店有 GOOD 信誉等级。因此，在结果表中，列出了 7 个客户名。

例 3.5

写一个 SQL 查询，列出 S_CUSTOMER 表中具有 EXCELLENT 信誉等级或与 sales-rep-id 为 12 的销售代表相关的所有客户名、信誉等级及销售代表。

```
SELECT name, credit_rating, sales_rep_id
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
OR sales_rep_id = '12';
```

结果表如下所示，只有两个商店与 EXCELLENT 信誉等级匹配，仅有两个商店与 sales_rep_id 为 12 的销售代表条件匹配。注意即使其余的商店同时满足了这两个条件，在输出的结果表中也只列出一次。OR 逻辑运算符是相容的，即它选择与任一条件或所有条件相匹配的元组。

```
NAME                      CREDIT_RA SAL
------------------------- --------- --
Sports,Inc                EXCELLENT 12
Athletics For All         EXCELLENT 12
Shoes for Sports          EXCELLENT 14
BJ Athletics              POOR      12
Great Athletes            EXCELLENT 12
Athletics Two             EXCELLENT 14
Athletes Attic            POOR      12
```

可以使用任意数量的 OR 条件，条件列出的顺序是无关紧要的。

例 3.6

写出一个 SQL 查询，列出在 S_CUSTOMER 表中具有 EXCELLENT 信誉等级或所在地区邮政编码为 22809，或与 sales_rep_id 为 12 的销售代表相关的所有客户名、信誉等级、邮政编码及销售代表。

```
SELECT name,credit_rating,zip_code,sales_rep_id
FROM s_customer
WHERE credit_rating = 'EXCELLENT'
OR zip_code = '22809'
OR sales_rep_id = '12';
```

在结果表中，只有商店名为 Athletic Attire 和 Athletics One 的未被列出。因为它们不满足任一条件。其中 Athletic Attire 具有以下值：信誉等级为 GOOD，邮政编码为 22808，销售代理的 id 为 14。

3.1.3 NOT 逻辑运算符

WHERE 子句通常告诉你在结果表中包含哪些元组。但有时你想指出被排除的元组。NOT 逻辑运算符用于指出被排除的列值。AND 及 OR 逻辑运算符可连接两个或两个以上的条件，是二目运算符，而 NOT 只能作用于一个条件，是一目运算符。在 WHERE 条件中，使用 NOT 的语法如下所示：

```
SELECT column_list
FROM table_name
WHERE NOT condition;
```

例 3.7

写出一个 SQL 查询，列出 S_CUSTOMER 表中邮政编码不为 22808 的全部客户名、邮政编码及销售代表 ID。

```
SELECT name, zip_code, sales_rep_id
FROM s_customer
WHERE NOT zip_code = '22808';
NAME                 ZIP_CODE        SAL
-------------------- --------------- ---
Sports, Inc          22809            12
Toms Sporting Goods  22809            14
Shoes for Sports     22809            14
BJ Athletics         22810            12
Athletics One        17601            14
Great Athletes       17602            12
Athletics Two        17602            14
Athletes Attic       17601            12
```

注意，条件中 NOT 的位置在属性或列名之前，读起来让人感到很别扭，用以下方式放置 NOT 可能会好一些：

```
WHERE zip_code NOT = '22808';
```

但将 NOT 逻辑运算符紧挨着等于关系运算符（=）是无效的，这样做会使 RDBMS 产

生一个错误信息。如果想以这种排序方式书写，可使用不等关系运算符（< >），有关说明见第 1 章。以下两个条件是可互换的，将产生同样的输出结果。

```
WHERE NOT zip_code = '22808';
WHERE zip_code < > '22808';
```

如前所述，NOT 只能作用于一个条件。如果要排除两项值，需重复使用 NOT 运算符。

例 3.8

写出一个 SQL 查询，列出 S_CUSTOMER 表中邮政编码不为 22808 同时也不与 sales rep_id 为 14 的销售代表相关的所有客户名、邮政编码及销售代表。

```
SELECT name, zip_code, sales_rep_id
FROM s_customer
WHERE NOT zip_code = '22808'
AND NOT sales_rep_id = '14';
NAME                 ZIP_CODE        SAL
-------------------- --------------- ---
Sports,Inc           22809           12
BJ Athletics         22810           12
Great Athletes       17602           12
Athletes Attic       17601           12
```

注意，NOT 及 AND 逻辑运算符的联合使用。WHERE 子句

```
WHERE NOT zip_code = '22808' AND sales_rep_id = '14';
```

将产生极为不同的结果，如下所示。NOT 是一个一目运算符，如果不用括号的话，必须对每一个条件重复使用 NOT。

```
NAME                 ZIP_CODE        SAL
-------------------- ---- ---------- ---
Toms Sporting Goods  22809           14
Shoes for Sports     22809           14
Athletics One        17601           14
Athletics Two        17602           14
```

当使用布尔运算符时，对一些表示同样条件的等值方式进行检查很有帮助。表 3-2 给出了联合使用 AND、OR、NOT 及括号表达同样条件的可供选择的方式，这些条件也称为德·摩根定律。

表 3-2 逻辑恒等式

a. NOT (p AND q)	等于	NOT p OR NOT q
b. NOT (p OR q)	等于	NOT p AND NOT q
c. NOT (NOT p)	等于	p

例如，检查在例 3.8 中的查询，其 WHERE 条件为：

```
WHERE NOT zip_code = '22808' AND NOT sales_rep_id = '14';
```

按照表 3-2 中的语句 a，可以按以下方式重新书写 WHERE 条件，得到同样的结果：

```
SELECT name, zip_code, sales_rep_id
FROM s_customer
WHERE NOT (zip_code = '22808' OR sales_rep_id = '14');
```

例 3.9 将讲解表 3-2 中的语句 b，注意 NOT 与 OR 一起使用的情况。如果 NOT 与 OR 一起使用在一个没有括号的条件里，由于未排除任何项，往往会列出整个表。

例 3.9

使用下列 SELECT 语句会得到什么结果？

```
SELECT name, zip_code
FROM s_customer
WHERE NOT zip_code = '22809'
OR NOT zip_code = '22808';
```

因为邮政编码为 22809 的元组满足条件 NOT zip_code = '22808'，其他邮政编码的元组满足条件 NOT zip_code = '22809'，因此将打印出整个表。此外，其他任一邮政编码均满足这两个条件，因此列出了整个表。下列替代形式会产生同样的结果

```
SELECT name, zip_code
FROM s_customer
WHERE NOT (zip_code = '22809' AND zip_code = '22808');
```

3.1.4　复合条件的优先级

只要仔细放置上述三个布尔运算符，就可联合使用在 WHERE 子句的任意复合条件中。优先级与其他程序设计语言相似。表 3-3 给出了布尔运算符计算的优先级及其真值表。可以使用圆括号修改运算的顺序。仔细检查下面的例子。正确的复合子句格式，并不总是该问题英语表达的反映。

表 3-3　布尔运算符的优先级

运算符	定义	说明	顺序
( )		在圆括号内要计算的东西	最先计算
NOT p	NOT p F T T F	将 p 值改变为相反的值	第二计算
p AND q	p AND q T T T T F F F F T F F F	条件 p 和条件 q 必须为真	第三计算
p OR q	p OR q T T T T T F F T T F F F	条件 p 或条件 q 任一或两者必须为真	最后计算

例 3.10

假设你想检索邮政编码为 22808 或 22809 且具有极好信誉等级的客户。第一种书写该查询的方式如下。会得到什么样的结果表?

```
SELECT name, zip_code, credit_rating
FROM s_customer
WHERE zip_code = '22809
OR zip_code = '22808'
AND credit_rating = 'EXCELLENT';
NAME                 ZIP_CODE        CREDIT_RA
-------------------- --------------- ---------
Sports,Inc           22809           EXCELLENT
Toms Sporting Goods  22809           POOR
Athletics For All    22808           EXCELLENT
Shoes for Sports     22809           EXCELLENT
```

输出结果不满足查询要求，这是因为首先计算了 AND。所有具极好信誉等级而且邮政编码为 22808 的行全被包括在内；然后，计算 OR，所有邮政编码为 22809 的元组被列出。为了检索所希望的元组，应该首先计算 OR。为了强调这个顺序，使用圆括号来检索在这两个邮政编码里的全部信息；然后，计算 AND，以选择所需要的元组。结果表产生去掉 Toms Sporting Goods 元组（例 3.10 中）的正确语句如下：

```
SELECT name, zip_code, credit_rating
FROM s_customer
WHERE (zip_code = '22809'OR zip_code = '22808')
AND credit_rating = 'EXCELLENT';
```

例 3.11

写出一个 SQL 语句，列出 S_CUSTOMER 表中具有任意信誉等级，但不与 sales_rep_id 为 12 的销售代表相关的所有客户名、邮政编码及销售代表，或者与任一销售代表相关，但没有 EXCELLENT 信誉等级客户的所有客户名、邮政编码及销售代表。

a. 回答上述的问题，第一次尝试如下：

```
SELECT name, credit_rating, sales_rep_id
FROM s_customer
WHERE NOT sales_rep_id = '12'
OR credit_rating = 'EXCELLENT';
NAME                 CREDIT_RA SAL
-------------------- --------- ---
Sports,Inc           EXCELLENT 12
Toms Sporting Goods  POOR      14
Athletic Attire      GOOD      14
```

```
Athletics For All       EXCELLENT 12
Shoes for Sports        EXCELLENT 14
Athletics One           GOOD      14
Great Athletes          EXCELLENT 12
Athletics Two           EXCELLENT 14
```

但这个查询列出了那些有 EXCELLENT 信誉等级的元组，而不是那些没有 EXCELLENT 信誉等级的元组。为了使 NOT 用于条件中的两个部分，必须使用圆括号。

b. 第二次尝试与上述第一次类似，如下所示。注意，结果表与第一个表有很大不同，但该尝试仍不能满足要求。

```
SELECT name, credit_rating, sales_rep_id
FROM s_customer
WHERE NOT (sales_rep_id = '12'
OR credit_rating = 'EXCELLENT');
NAME                    CREDIT_RA SAL
----------------------- --------- ---
Toms Sporting Goods     POOR      14
Athletic Attire         GOOD      14
Athletics One           GOOD      14
```

此表列出了那些既不与 Id 为 12 的销售代表相关，也没有 EXCELLENT 信誉等级的客户。但它漏列了那些与 Id 为 12 的销售代表相关，但没有 EXCELLENT 信誉等级的客户以及那些不与 Id 为 12 的销售代理相关但有任一信誉等级的客户。

c. 本题原来的要求是列出在 S_CUSTOMER 表中具有任意信誉等级，但不与 sales_rep_id 为 12 的销售代理相关的所有客户，或者与任一销售代理相关，但没有 EXCELLENT 信誉等级的所有客户。它应包括与 Id 为 12 的销售代理相关且信誉等级为 POOR 或 GOOD 的所有客户。第三次尝试能满足要求。

```
SELECT name, credit_rating, sales_rep_id
FROM s_customer
WHERE NOT sales_rep_id = '12'
OR NOT credit_rating = 'EXCELLENT';
NAME                    CREDIT_RA SAL
----------------------- --------- ---
Toms Sporting Goods     POOR      14
Athletic Attire         GOOD      14
Shoes for Sports        EXCELLENT 14
BJ Athletics            POOR      12
Athletics One           GOOD      14
Athletics Two           EXCELLENT 14
Athletes Attic          POOR      12
```

例 3.12

写出一个 SQL 查询，显示除了那些邮政编码为 22809 且具有 EXCELLENT 信誉等级以

外的所有客户的名称、邮政编码及信誉等级。

为了构建该查询，首先建立要排除的组的条件（zip_code = '22809' AND credit_rating = 'EXCELLENT'），然后将 NOT 置于该条件之前

```
SELECT name, zip_code, credit_rating
FROM s_customer
WHERE NOT(zip_code = '22809' AND credit_rating = 'EXCELLENT');
NAME                 ZIP_CODE        CREDIT_RA
-------------------- --------------- ---------
Toms Sporting Goods  22809           POOR
Athletic Attire      22808           GOOD
Athletics For All    22808           EXCELLENT
BJ Athletics         22810           POOR
Athletics One        17601           GOOD
Great Athletes       17602           EXCELLENT
Athletics Two        17602           EXCELLENT
Athletes Attic       17601           POOR
```

注意，并非排除邮政编码为 22809 的所有客户，也并非排除具 EXCELLENT 信誉等级的所有客户。在设计任何复合条件时都必须多加小心。

3.2 字符匹配——LIKE 子句及通配符

前面所有检索元组的例子都是基于一列或 n 列的精确值。但有时你并不知道精确值，或你想检索具有相似值的元组。LIKE 逻辑运算符用于匹配指定的样式。例如，你想检索那些人名以 J 开头的元组或那些居住在 St 而不是 Rd 或 Blvd 的元组。SQL 为字符型的列提供了一个字符匹配机制。LIKE 逻辑运算符只能对字符串列发挥作用。可将 NOT、AND 及 OR 与 LIKE 组合，一起构成复合条件。LIKE 逻辑运算符的语法如下所示：

```
SELECT column_list
FROM table_name
WHERE column_name LIKE 'pattern';
```

包括在单引号内的匹配串可以利用通配符百分号（%）及下划线（_）。表 3-4 是包含了较多信息的客户表的子集，下一节将把它作为例子使用。

表 3-4 扩展的客户表

ID	NAME	ADDRESS	CITY	ST	ZIP_COD	PHONE
301	Sports,Inc	72 High St	Harrisonburg	VA	22809	540-123-4567
302	Toms Sporting Goods	6741 Main St	Harrisonburg	VA	22809	540-987-6543
303	Athletic Attire	54 Market St	Harrisonburg	VA	22808	540-123-6789
304	Athletics For All	286 Main St	Harrisonburg	VA	22808	540-987-1234
305	Shoes for Sports	538 High St	Harrisonburg	VA	22809	540-123-9876
306	BJ Athletics	632 Water St	Harrisonburg	VA	22810	540-987-9999
403	Athletics One	912 Columbia Rd	Lancaster	PA	17601	717-234-6786
404	Great Athletes	121 Litiz Pike	Lancaster	PA	17602	717-987-2341
405	Athletics Two	435 High Rd	Lancaster	PA	17602	717-987-9875
406	Athletes Attic	101 Greenfield Rd	Lancaster	PA	17601	717-234-9888

3.2.1 百分号（%）通配符

百分号可被用作代表零个或多个泛指字符的字符串。例如‘S%’表示以大写字母S开头的任意长度（包括单一字母S）的字符串列值；‘%s’表示以小写字母s结束的任意长度的字符串列值，该值必须以s结束且其后不含空格符；匹配串‘%s%’表示任意位置具有小写s的单词。记住在引号内的匹配串区分字母的大小写。

例3.13

写出一个SQL查询，列出客户名以Athlete或Athletics等单词开头的客户的名称、地址、所在城市及州名。将它们按客户名的字母顺序列出。

```
SELECT name, address, city, state
FROM s_customer
WHERE name LIKE 'Athl%'
ORDER BY name;
```

NAME	ADDRESS	CITY	STATE
Athletes Attic	101 Greenfield Rd	Lancaster	PA
Athletic Attire	54 Market St	Harrisonburg	VA
Athletics For All	286 Main St	Harrisonburg	VA
Athletics One	912 Columbia Rd	Lancaster	PA
Athletics Two	435 High Rd	Lancaster	PA

例3.14

写出一个SQL查询，列出居住在street而非road上的客户的名称、地址、所在城市及州。将它们按客户名的字母顺序列出。

```
SELECT name, address, city, state
FROM s_customer
WHERE address LIKE '%St'
ORDER BY name;
```

NAME	ADDRESS	CITY	STATE
Athletic Attire	54 Market St	Harrisonburg	VA
Athletics For All	286 Main St	Harrisonburg	VA
BJ Athletics	632 Water St	Harrisonburg	VA
Shoes for Sports	538 High St	Harrisonburg	VA
Sports, Inc	72 High St	Harrisonburg	VA
Toms Sporting Goods	6741 Main St	Harrisonburg	VA

例 3.15

写出一个SQL查询，列出不住在street上的客户的名称、地址、所在城市及州，没有规定列出顺序。

```
SELECT name, address, city, state
FROM s_customer
WHERE address NOT LIKE '%St';
```

NAME	ADDRESS	CITY	STATE
Athletics One	912 Columbia Rd	Lancaster	PA
Great Athletes	121 Litiz Pike	Lancaster	PA
Athletics Two	435 High Rd	Lancaster	PA
Athletes Attic	101 Greenfield Rd	Lancaster	PA

注意单词NOT所处的位置。当它与LIKE同时使用时，NOT按正规英语语序放置。

例 3.16

写出一个SQL查询，列出不管地址是street还是road，只要地址中有High的客户的名称、地址、所在城市及州。

```
SELECT name, address, city, state
FROM s_customer
WHERE address LIKE '%High%';
```

NAME	ADDRESS	CITY	STATE
Sports,Inc	72 High St	Harrisonburg	VA
Shoes for Sports	538 High St	Harrisonburg	VA
Athletics Two	435 High Rd	Lancaster	PA

该匹配串与地址中有High的任意记录相匹配。

3.2.2 下划线通配符

下划线（_）通配符常用于代表任意一个字符。当你知道字符类型列的准确长度，并需要一个通配符来指明字符数时，将其与字符列一起使用。

例 3.17

写出一个SQL查询，列出美国地区代码为540的所有客户的名称及电话号码。电话号码的长度为定长，前三位数为地区代码，中间三位数为交换台号，最后还有四位数，格式为XXX-XXX-XXXX。由于你知道该列的准确长度，因此可使用一个下划线通配符来代表电话号码中的任一位数字。

```
SELECT name, phone
FROM s_customer
WHERE phone LIKE '540-___-____';
NAME                      PHONE
------------------------  ------------
Sports,Inc                540-123-4567
Toms Sporting Goods       540-987-6543
Athletic Attire           540-123-6789
Athletics For All         540-987-1234
Shoes for Sports          540-123-9876
BJ Athletics              540-987-9999
```

注意，在此例中，你也可以用匹配串‘540%’得到同样的结果。但有时要求的串长度不同会使这两种表达结果产生一些差异。

例 3.18

假设有两个特许店 Athletics One 和 Athletics Two。写出仅列出这两个客户的名称及电话号码的 SQL 查询。

```
SELECT name, phone
FROM s_customer
WHERE name LIKE 'Athletics ___ ;
```

该匹配串包含单词 Athletics，后面紧跟一个空格及三个下划线字符，这个查询会产生所想要的结果。此时，如果你用‘Athletics%’，你还会检索到客户 Athletics for All 的电话号码。

例 3.19

写出一个 SQL 查询，列出不限地区代码但具有 987 交换台号（中间三位数）的所有客户的名称及电话号码。由于知道电话号码的准确长度（三位数 - 三位数 - 四位数），因此可以使用一个下划线通配符来代表电话号码中其余的每一位数。

```
SELECT name, phone
FROM s_customer
WHERE phone LIKE '___-987-____';
NAME                      PHONE
------------------------  ------------
Toms Sporting Goods       540-987-6543
Athletics For All         540-987-1234
BJ Athletics              540-987-9999
Great Athletes            717-987-2341
Athletics Two             717-987-9875
```

结果表证明该查询使用了下划线通配符，检索到了正确的元组。但如果使用'%987%'就会得到下面的表，它含有其他元组，它与列中任何地方出现的987均匹配，但并不正好是在所要求的位置上。

```
NAME                    PHONE
----------------------  ------------
Toms Sporting Goods     540-987-6543
Athletics For All       540-987-1234
Shoes for Sports        540-123-9876
BJ Athletics            540-987-9999
Great Athletes          717-987-2341
Athletics Two           717-987-9875
```

百分号通配符用于与任意字符组的匹配，而下划线通配符则与任一个字符匹配。若与一个百分号通配符或下划线通配符自身匹配，则会出现问题。为了表明你确实想匹配该通配符，可在查询的符号前面使用换码字符"\"。虽然这样做是有帮助的，但在许多SQL的实现中，它并不起作用。

3.3 匹配列表中的值或范围值

LIKE条件常用于匹配指定的字符串。有时，你想根据某个范围的值或某个指定列表中的值来选择行。这时，可使用BETWEEN及IN运算符完成这些功能。通常可以使用布尔运算符AND或OR来重复使用BETWEEN和IN的查询，但有时使用BETWEEN和IN会更方便。

3.3.1 BETWEEN运算符

用BETWEEN标志可接受值的范围。为表示一个范围，必须指出一个低限值和一个高限值，并且应先指出低限值。被选择的值的范围包括低限值、高限值以及位于它们之间的任意一个值，语法如下：

```
SELECT column_list
FROM table_name
WHERE column_name BETWEEN lowvalue AND highvalue;
```

例3.20

写出一个SQL查询，列出客户ID在303及306之间的客户的ID、名字及电话号码。

```
SELECT id, name, phone
FROM s_customer
WHERE id BETWEEN '303' AND '306';
```

该查询检索到 id 为 303、304、305 及 306 的任意元组，所得到的结果如下：

```
ID   NAME                 PHONE
---  -------------------  ------------
303  Athletic Attire      540-123-6789
304  Athletics For All    540-987-1234
305  Shoes for Sports     540-123-9876
306  BJ Athletics         540-987-9999
```

获得同样结果的另一种查询方式如下：

```
SELECT id, name, phone
FROM s_customer
WHERE id >= '303' AND id <= '306';
```

注意，BETWEEN 的使用可使语句说明得更清楚。

BETWEEN 运算符可与数值或字符串列一起使用，下面例子对另一个字符串列进行了说明。

例 3.21

写出一个 SQL 查询，列出客户的 ID、名字及电话号码，要求客户名字的开头是从 A 到 G 之间的字母。

a. 尝试设计的第一个查询如下：

```
SELECT id, name, phone
FROM s_customer
WHERE name BETWEEN 'A' AND 'G';
ID   NAME                 PHONE
---  -------------------  ------------
303  Athletic Attire      540-123-6789
304  Athletics For All    540-987-1234
306  BJ Athletics         540-987-9999
403  Athletics One        717-234-6786
405  Athletics Two        717-987-9875
406  Athletes Attic       717-234-9888
```

b. 注意在结果表中不包括 Great Athletes 商店。这是因为指明的高限值是 'G'，即不允许 'G' 后还有字母。为了在结果表中包括 Great Athletes，可以用 'H' 作为高限值。如果有真实名称为 'H' 的商店，则它也被包括在内，但实际上没有。正确的查询应该如下：

```
SELECT id, name, phone
FROM s_customer
WHERE name BETWEEN 'A' AND 'H';
```

例 3.22

写出一个 SQL 查询，列出客户 ID 不在 302 至 306 之间的所有客户的 ID、名字及电话

号码。使用布尔运算符 NOT 排除 ID 在 302 至 306 之间的元组

```
SELECT id, name, phone
FROM s_customer
WHERE id NOT BETWEEN '302' AND '306';
ID   NAME                PHONE
---  ----------------    ------------
301  Sports,Inc          540-123-4567
403  Athletics One       717-234-6786
404  Great Athletes      717-987-2341
405  Athletics Two       717-987-9875
406  Athletes Attic      717-234-9888
```

3.3.2 IN运算符

使用 IN 运算符，依据某个列表中的值选择一行，列表中值的个数可以任意多。值在圆括号内加以说明，各项之间以逗号隔开。语法如下：

```
SELECT column_list
FROM table_name
WHERE column_name IN (value_1, value_2, . . . value_n);
```

该查询可检索到在某列中具有任一指定值的行。通常，将指定的各项按一定顺序列出有助于提高可读性，但对这一点并无要求。这些项可为数值或字符串，但字符串的各项必须置于单引号内。

例 3.23

写出查询客户的 ID 为 303、305、403 及 406 的客户的 ID、名字及电话号码的 SQL 语句

```
SELECT id, name, phone
FROM s_customer
WHERE id IN ('303', '305', '403', '406');
```

只有 Athletic Attire、Shoes for Sports、Athletics One 及 Athletes Attic 这 4 个元组被选择。

例 3.24

写出一个 SQL 语句，查询客户的 ID 不为 303、305、403 及 406 的客户 ID、名字及电话号码。加上 NOT 运算符变反该查询。

```
SELECT id, name, phone
FROM s_customer
WHERE id NOT IN ('303', '403', '305', '406');
```

只有 Athletic Attire、Shoes for Sports、Athletics One 及 Athletes Attic 这 4 个元组未被选择。在圆括号中的值可以任何顺序写出

例 3.25

写出查询具有 EXCELLENT 或 POOR 信誉等级的客户名字及信誉等级的 SQL 语句。

```
SELECT name, credit_rating
FROM s_customer
WHERE credit_rating IN ('EXCELLENT', 'POOR');
```

结果表将列出在这两个范畴内的所有客户

问题与答案

注意，在回答下列问题之前，重新运行运动用品数据库脚本来刷新数据库。同时，使用雇员表 S_EMP 的下列子集回答问题

ID	LAST_NAME	FIRST_NAME	MAN	TITLE	DEP	SALARY
1	Martin	Carmen		President	50	4500
2	Smith	Doris	1	VP, Operations	41	2450
3	Norton	Michael	1	VP, Sales	31	2400
4	Quentin	Mark	1	VP, Finance	10	2450
5	Roper	Joseph	1	VP, Administration	50	2550
6	Brown	Molly	2	Warehouse Manager	41	1600
7	Hawkins	Roberta	2	Warehouse Manager	42	1650
8	Burns	Ben	2	Warehouse Manager	43	1500
9	Catskill	Antoinette	2	Warehouse Manager	44	1700
10	Jackson	Marta	2	Warehouse Manager	45	1507
11	Henderson	Colin	3	Sales Representative	31	1400
12	Gilson	Sam	3	Sales Representative	32	1490
13	Sanders	Jason	3	Sales Representative	33	1515
14	Dameron	Andre	3	Sales Representative	35	1450
15	Hardwick	Elaine	6	Stock Clerk	41	1400
16	Brown	George	6	Stock Clerk	41	940
17	Washington	Thomas	7	Stock Clerk	42	1200
18	Patterson	Donald	7	Stock Clerk	42	795
19	Bell	Alexander	8	Stock Clerk	43	850
20	Gantos	Eddie	9	Stock Clerk	44	800
21	Stephenson	Blaine	10	Stock Clerk	45	860
22	Chester	Eddie	9	Stock Clerk	44	800
23	Pearl	Roger	9	Stock Clerk	34	795
24	Dancer	Bonnie	7	Stock Clerk	45	860
25	Schmitt	Sandra	8	Stock Clerk	45	1100

3.1 写出一个SQL查询，显示在部门41工作、工资高于1000美元的所有雇员的姓氏、名字、部门ID及工资。

```
SELECT last_name, first_name, dept_id, salary
FROM s_emp
WHERE dept_id = '41'AND salary > 1000;
LAST_NAME        FIRST_NAME          DEP    SALARY
---------------  ---  -------------  ------ ------
Smith            Doris               41     2450
Brown            Molly               41     1600
Hardwick         Elaine              41     1400
```

George Brown在部门41工作，但他的工资低于1000美元，因此他未被列入表中。记住当使用AND时，两个条件必须都为真。

3.2 写出一个SQL查询，显示在部门42至部门44工作、工资高于1600美元的所有仓库经理的姓氏、名字、部门ID及工资。

```
SELECT last_name, first_name, dept_id, salary
FROM s_emp
WHERE title = 'Warehouse Manager'
AND dept_id >= '42'
AND dept_id <= '44'
AND salary > 1600;
LAST_NAME        FIRST_NAME          DEP    SALARY
---------------  ---------------     -----  ------
Hawkins          Roberta             42     1650
Catskill         Antoinette          44     1700
```

记住字符串值区分大小写，因此在引号中的职务名的大小写必须正确。如将条件写成title = 'WAREHOUSE MANAGER'，会产生一个空表。同样为了检索在42～44之间的所有部门，部门ID必须大于等于42，且小于等于44。部门43的仓库经理由于工资低于1600美元，故未被列入表中。

3.3 写出一个SQL查询，显示在部门31及在部门41工作的雇员的姓名、部门ID及工资。

```
SELECT last_name, first_name, dept_id
FROM s_emp
WHERE dept_id = '31'
OR dept_id = '41';
LAST_NAME        FIRST_NAME          DEP
---------------  ---------------     ---
Smith            Doris               41
Norton           Michael             31
Brown            Molly               41
Henderson        Colin               31
Hardwick         Elaine              41
Brown            George              41
```

注意，题目中虽然用的是部门31及部门41，但这个查询不能用AND，因为部门ID值不可能既是31，又是41。因此，我们使用OR来查询在部门31或部门41的雇员

3.4 写出一个SQL查询，显示经理ID为1、2或NULL的所有雇员的姓氏、名字、职务及其经理ID，将结果按职务、姓氏、名字的字母顺序排序。记住：你可按多个需要的属性排序。

```
SELECT title, last_name, first_name, manager_id
FROM s_emp
WHERE manager_id is NULL
OR manager_id = '1'
OR manager_id = '2'
ORDER BY title, last_name, first_name;
```

```
TITLE                LAST_NAME    FIRST_NAME    MAN
-------------------  -------- ---  ----------  -----------
President            Martin       Carmen
VP, Administration   Roper        Joseph       1
VP, Finance          Quentin      Mark         1
VP, Operations       Smith        Doris        1
VP, Sales            Norton       Michael      1
Warehouse Manager    Brown        Molly        2
Warehouse Manager    Burns        Ben          2
Warehouse Manager    Catskill     Antoinette   2
Warehouse Manager    Hawkins      Roberta      2
Warehouse Manager    Jackson      Marta        2
```

为了检索有NULL值的经理ID，我们使用manager_id IS NULL，而没有用manager_id = NULL。记住：等号与并不IS一样。

3.5 写出一个SQL查询，显示工资不超过1200美元的所有雇员的姓氏、名字、部门ID及工资

```
SELECT last_name, first_name, dept_id, salary
FROM s_emp
WHERE NOT salary > 1200;
```

```
LAST_NAME        FIRST_NAME          DEP    SALARY
---------------  ----------------  -----  ------
Brown            George              41     940
Washington       Thomas              42    1200
Patterson        Donald              42     795
Bell             Alexander           43     850
Gantos           Eddie               44     800
Stephenson       Blaine              45     860
Chester          Eddie               44     800
Pearl            Roger               34     795
Dancer           Bonnie              45     860
Schmitt          Sandra              45    1100
```

在WHERE子句中的条件也可写为 WHERE salary < = 1200 如果问题中指定“工资不高于或不等于1200美元的所有雇员”，则条件应写为 WHERE NOT salary > = 1200，此时，在结果表中将不包括 Thomas Washington。注意，在这个查询中数值中间未用逗号。

3.6 写出一个SQL查询，并显示不在部门31或41工作，但工资高于2000美元的所有雇员的姓氏、名字、部门ID及工资的结果表。

```
SELECT last_name, first_name, dept_id, salary
FROM s_emp
WHERE salary > 2000
AND NOT dept_id = '31'
AND NOT dept_id = '41';
```

```
LAST_NAME          FIRST_NAME          DEP      SALARY
---------------    ---------------     -------  -- ---
Martin             Carmen              50       4500
Quentin            Mark                10       2450
Roper              Joseph              50       2550
```

记住，NOT的正确位置是放在列名前，问题中指定“不在部门31或41”意味着既不在部门31，也不在部门41，要小心使用条件中的单词。

3.7 写出一个SQL查询，并显示的结果表在部门41工作，是库存管理员或仓库经理的所有人员的姓氏、名字、职位及部门ID的结果表。

```
SELECT last_name, first_name, title, dept_id
FROM s_emp
WHERE dept_id = '41'
AND (title = 'Stock Clerk' OR title = 'Warehouse Manager');
```

```
LAST_NAME          FIRST_NAME          TITLE                DEP
---------------    -----  ---------    -----------------    ---
Brown              Molly               Warehouse Manager    41
Hardwick           Elaine              Stock Clerk          41
Brown              George              Stock Clerk          41
```

首先计算OR运算符。Doris Smith不包括在表中，因为她不满足职务条件。记住：如果没有圆括号，就会首先计算AND，列出的仓库经理不仅限于部门41。所得到的不正确的表如下所示：

```
LAST_NAME          FIRST_NAME          TITLE                DEP
-----  --------    ---------------     -----------------    ---
Brown              Molly               Warehouse Manager    41
Hawkins            Roberta             Warehouse Manager    42
Burns              Ben                 Warehouse Manager    43
Catskill           Antoinette          Warehouse Manager    44
Jackson            Marta               Warehouse Manager    45
Hardwick           Elaine              Stock Clerk          41
Brown              George              Stock Clerk          41
```

3.8　写出一个 SQL 查询，并显示工资高于 1500 美元，且不为仓库经理或不在部门 50 工作的所有人员的姓氏、名字、职务、部门 ID 及工资。

```
SELECT last_name, first_name, title, dept_id, salary
FROM s_emp
WHERE salary > 1500
AND NOT (title='Warehouse Manager' OR dept_id = '50');
```

```
LAST_NAME     FIRST_NAME   TITLE                DEP  SALARY
------------- ------------ -------------------- ---  ------
Smith         Doris        VP, Operations       41   2450
Norton        Michael      VP, Sales            31   2400
Quentin       Mark         VP, Finance          10   2450
Sanders       Jason        Sales Representative 33   1515
```

3.9　写出一个 SQL 查询，并显示姓氏以字母 B 开头的所有雇员的姓氏和名字，将他（她）们按姓氏、名字的字母顺序列出

```
SELECT last_name, first_name
FROM s_emp
WHERE last_name LIKE 'B%'
ORDER BY last_name, first_name;
```

```
LAST_NAME            FIRST_NAME
-------------------- --------------------
Bell                 Alexander
Brown                George
Brown                Molly
Burns                Ben
```

注意，姓氏是按字母顺序排列的，对于姓氏相同的两个雇员，是按名字的字母顺序排列的。

3.10　写出一个 SQL 查询，并显示姓氏以 son 结束的所有雇员的姓氏、名字及职务，将他（她）们按姓氏的字母顺序列出。

```
SELECT last_name, first_name, title
FROM s_emp
WHERE last_name LIKE '%son'
ORDER BY last_name;
```

```
LAST_NAME       FIRST_NAME      TITLE
--------------- --------------- --------------------
Gilson          Sam             Sales Representative
Henderson       Colin           Sales Representative
Jackson         Marta           Warehouse Manager
Patterson       Donald          Stock Clerk
Stephenson      Blaine          Stock Clerk
```

3.11　写出 SQL 查询，并显示可列出名字中包含字母 m 的所有雇员的名字及姓氏的结果表。

```
SELECT first_name, last_name
FROM s_emp
WHERE first_name LIKE '%m% ;
FIRST_NAME              LAST_NAME
--------------------    --------------------
Carmen                  Martin
Sam                     Gilson
Thomas                  Washington
```

结果表中未列出 Mark 、Molly 及名字中有大写字母 M 的雇员。为了包含 M 及 m 这两个字母，可在条件子句中使用 OR，如下所示，这个语句可检索名字中有大写或小写 m 的雇员元组。

```
SELECT first_name, last_name
FROM s_emp
WHERE first_name LIKE '%m%' OR first_name LIKE '%M%';
```

3.12 写出 SQL 查询，并显示可列出名字以 e 结尾，且名字正好为 6 个字母的所有雇员的名字、姓氏的结果表。

```
SELECT first_name, last_name
FROM s_emp
WHERE first_name LIKE '_____e';
FIRST_NAME              LAST_NAME
--------------------    --------------------
Elaine                  Hardwick
George                  Brown
Blaine                  Stephenson
Bonnie                  Dancer
```

注意，像 “Eddie” 等名字未被包括在内，因为条件中要求正好为 6 个字符。

3.13 写出 SQL 查询，并显示可列出名字以 M 开始，且名字正好为 5 个字母的所有雇员的名字、姓氏的结果表。

```
SELECT first_name, last_name
FROM s_emp
WHERE first_name LIKE 'M____';
FIRST_NAME              LAST_NAME
--------------------    --------------------
Molly                   Brown
Marta                   Jackson
```

再重复一遍，象 Mark 及 Michael 等名字未被包括在内，因为条件中要求正好为 5 个字符。

3.14 下划线通配符可用于匹配任意字符且长度正好的字符串，写出 SQL 查询，并显示可列出名字、姓氏都正好有 5 个字符的所有雇员的名字及姓氏的结果表。

```
SELECT first_name, last_name
FROM s_emp
WHERE first_name LIKE '_____' AND last_name LIKE '_____';
FIRST_NAME           LAST_NAME
-------------------- --------- --------
Doris                Smith
Molly                Brown
Roger                Pearl
```

3.15　写出 SQL 查询，并显示可列出在职务中没有空格字符的所有雇员的名字、姓氏及职位的结果表。

```
SELECT first_name, last_name, title
FROM s_emp
WHERE title NOT LIKE '% %';
```

所列出的惟一记录为 President、Carmen Martin。因为该职务只有一个单词，因此中间没有空格。

3.16　写出 SQL 查询，并显示可列出工资在 1 000 美元至 1 500 美元之间的所有雇员的名字、姓氏及工资的结果表。

```
SELECT first_name, last_name, salary
FROM s_emp
WHERE salary BETWEEN 1000 AND 1500;
FIRST_NAME           LAST_NAME            SALARY
-------------------- -------------------- ---------
Ben                  Burns                1500
Colin                Henderson            1400
Sam                  Gilson               1490
Andre                Dameron              1450
Elaine               Hardwick             1400
Thomas               Washington           1200
Sandra               Schmitt              1100
```

注意，Ben Burns 被包括在结果表中，因为他的工资正好为 1500 美元。记住在 SQL 查询中，既不可用逗号（即 1,500），也不可用美元符号。为了以这种方式格式化结果，可参看第 4 章。

3.17　写出 SQL 查询，结果表可列出用户 ID 以字母 p 至 s 开始的所有雇员的 ID、名字、姓氏及用户 ID。

```
SELECT id, first_name, last_name, userid
FROM s_emp
WHERE userid BETWEEN 'p' AND 't';
ID  FIRST_NAME           LAST_NAME            USERID
--- -------------------- -------------------- --------
18  Donald               Patterson            patterdv
23  Roger                Pearl                pearlrg
4   Mark                 Quentin              quentiml
5   Joseph               Roper                roperjm
13  Jason                Sanders              sanderjk
25  Sandra               Schmitt              schmitss
2   Doris                Smith                smithdj
21  Blaine               Stephenson           stephebs
```

记住：为了包括所有以 s 开始的用户 ID，必须以字符 't' 作为上限。

3.18 写出 SQL 查询，结果表可列出雇员 ID 号为 13、15、17 及 10 的所有雇员 ID、名字、姓氏（记住，在圆括号中所列出的值可为任意顺序）。

```
SELECT id, first_name, last_name
FROM s_emp
WHERE id IN ('13' , '15', '17', '10');
ID  FIRST_NAME           LAST_NAME
--- -------------------- --------------------
10  Marta                Jackson
17  Thomas               Washington
15  Elaine               Hardwick
13  Jason                Sanders
```

可用 ORDER BY 子句将输出结果按 ID 顺序显示。

3.19 写出 SQL 查询，结果表可列出姓氏为 Brown、Burns、Washington 及 Jackson 的所有雇员的 ID、名字、姓氏及职务，将他们按字母顺序显示。

```
SELECT id, first_name, last_name, title
FROM s_emp
WHERE last_name IN ('Brown', 'Burns', 'Washington', 'Jackson')
ORDER BY last_name;
ID  FIRST_NAME    LAST_NAME            TITLE
--- ------------- -------------------- ------------------
6   Molly         Brown                Warehouse Manager
16  George        Brown                Stock Clerk
8   Ben           Burns                Warehouse Manager
10  Marta         Jackson              Warehouse Manager
17  Thomas        Washington           Stock Clerk
```

3.20 写出 SQL 查询，结果表可列出副总裁的姓氏、名字、他们的部门 ID 及部门名称。

```
SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp, s_dept
WHERE s_emp.dept_id = s_dept.id AND title LIKE 'VP%';
LAST_NAME            DEP NAME
-------------------- --- --------------------
Smith                41  Operations
Norton               31  Sales
Quentin              10  Finance
Roper                50  Administration
```

注意，这个查询需要来自两个不同表的数据，因此为清楚起见，表名必须放在每一列名前面。

3.21 写出 SQL 查询，结果表可列出姓氏以 H 开头的所有销售代表的客户名称及该销售代表的姓氏。

```
SELECT s_customer.name, s_emp.last_name
FROM s_emp, s_customer
WHERE s_emp.id = s_customer.sales_rep_id
AND s_emp.last_name LIKE 'H%';
NAME                 LAST_NAME
-------------------- ---------------
New Delhi Sports     Henderson
Ladysport            Henderson
Kim's Sporting Goods Henderson
Beisbol Si!          Henderson
Helmut's Sports      Henderson
Sports Emporium      Henderson
Sports Retail        Henderson
Sports Russia        Henderson
```

该查询需要来自雇员表及客户表的信息

3.22 写出SQL查询，在结果表中可列出所有具有POOR信誉等级的客户的名称、地域名称、销售代表的姓氏和国家名称（提示：你需要查询三个不同的表）。

```
SELECT s_customer.name "CUSTOMER", s_region.name "REGION",
     s_emp.last_name "SALES REP", s_customer.country 'COUNTRY"
FROM s_customer, s_region, s_emp
WHERE s_customer.sales_rep_id = s_emp.id
AND s_customer.region_id = s_region.id
AND s_customer.credit_rating = 'POOR';
CUSTOMER             REGION        SALES REP   COUNTRY
-------------------- ------------- ----------- -----------
Toms Sporting Goods  North America Dameron     US
BJ Athletics         North America Gilson      US
Athletes Attic       North America Gilson      US
Sports Retail        North America Henderson   US
Sports Russia        Europe        Henderson   Russia
```

补 充 题

注意：在回答下列问题之前，重新运行SG脚本来刷新数据库。同时，使用S_ORD定单表回答3.23至3.36之间的补充题。

```
ID  DATE_ORDE DATE_SHIP SAL   TOTAL PAYMEN
--- --------- --------- --- ------- ------
100 31-AUG-92 10-SEP-92 11   601100 CREDIT
101 31-AUG-92 15-SEP-92 14   8056.6 CREDIT
102 01-SEP-92 08-SEP-92 15     8335 CREDIT
103 02-SEP-92 22-SEP-92 15      377 CASH
104 03-SEP-92 23-SEP-92 15    32430 CREDIT
105 04-SEP-92 18-SEP-92 11  2722.24 CREDIT
```

```
106 07-SEP-92 15-SEP-92 12    15634 CREDIT
107 07-SEP-92 21-SEP-92 15   142171 CREDIT
108 07-SEP-92 10-SEP-92 13   149570 CREDIT
109 08-SEP-92 28-SEP-92 11  1020935 CREDIT
110 09-SEP-92 21-SEP-92 11  1539.13 CASH
111 09-SEP-92 21-SEP-92 11     2770 CASH
97  28-AUG-92 17-SEP-92 12    84000 CREDIT
98  31-AUG-92 10-SEP-92 14      595 CASH
99  31-AUG-92 18-SEP-92 14     7707 CREDIT
112 31-AUG-92 10-SEP-92 12      550 CREDIT
```

3.23 写出 SQL 查询，在结果表中可列出销售代表为 14，订货日期为 1992.8.31 的订货单的订单 ID、订货日期、销售代表及订货总金额。

3.24 写出 SQL 查询，在结果表中可列出销售代表为 14，订货日期为 1992.8.31 且订货总金额大于 6000 美元订单的订单 ID、订货日期、销售代表及订货总金额。

3.25 写出 SQL 查询，在结果表中可列出订货日期为 1992.8.31 或订货单以现金方式付款的订单 ID、订货日期、销售代表及支付方式。

3.26 写出 SQL 查询，在结果表中可列出所有订货日期不为 1992.8.31 的订单 ID、订货日期、销售代表及支付方式。

3.27 写出 SQL 查询，在结果表中可列出所有订货日期不为 1992.8.31 的订单 ID、订货日期、销售代表及支付方式。

3.28 写出 SQL 查询，在结果表中可列出订货日期为 1992.8.31 或 1992.9.1，销售代表为 12 的所有订单 ID、订货日期、销售代理及订货总金额。

3.29 写出 SQL 查询，在结果表中可列出订货日期在 8 月底之前或装运日期在 9 月 10 日之前的订单 ID、订货日期及装运日期。

3.30 写出 SQL 查询，在结果表中可列出不包括装货日期在 9 月 15 日之后且以信用卡方式付款的其他订单 ID、装运日期及付款方式。结果按装运日期进行排序。

3.31 写出 SQL 查询，在结果表中可列出订货总金额超过 6000 美元，且订货日期不在 8 月 31 之后或装运日期不在 9 月 10 日之前的订单 ID、订货日期、装货日期及订货总金额

3.32 写出 SQL 查询，在结果表中可列出订货日期在 8 月份的订单 ID、订货日期、装运日期及订货总金额

3.33 写出 SQL 查询，结果表可列出订货总金额在 6000 至 8000 美元之间的订单 ID、订货日期、销售代表及订货总金额。

3.34 写出 SQL 查询，结果表可列出订货日期在 1992 年 9 月的订单 ID、订货日期及装运日期。

3.35 写出 SQL 查询，结果表可列出客户 ID 为 204、206、201 及 202 的订单 ID、客户 ID 及订货日期。将结果表按客户 ID 排序。

3.36 写出 SQL 查询，结果表可列出所有订货单的客户名称、客户 ID 及支付方式。将结果

表按客户 ID 排序。任何客户都不要显示两次 （提示：该查询需要两个不同的表）

使用 WORLD_CITIES 表的子集回答补充题 3.37 至 3.42。

CITY	COUNTRY	CONTINENT
ATHENS	GREECE	EUROPE
ATLANTA	UNITED STATES	NORTH AMERICA
DALLAS	UNITED STATES	NORTH AMERICA
NASHVILLE	UNITED STATES	NORTH AMERICA
VICTORIA	CANADA	NORTH AMERICA
PETERBOROUGH	CANADA	NORTH AMERICA
VANCOUVER	CANADA	NORTH AMERICA
TOLEDO	UNITED STATES	NORTH AMERICA
WARSAW	POLAND	EUROPE
LIMA	PERU	SOUTH AMERICA
RIO DE JANEIRO	BRAZIL	SOUTH AMERICA
SANTIAGO	CHILE	SOUTH AMERICA
BOGOTA	COLOMBIA	SOUTH AMERICA
BUENOS AIRES	ARGENTINA	SOUTH AMERICA
QUITO	ECUADOR	SOUTH AMERICA
CARACAS	VENEZUELA	SOUTH AMERICA
MADRAS	INDIA	ASIA
NEW DELHI	INDIA	ASIA
BOMBAY	INDIA	ASIA
MANCHESTER	ENGLAND	EUROPE
LONDON	ENGLAND	EUROPE
MOSCOW	RUSSIA	EUROPE
PARIS	FRANCE	EUROPE
SHENYANG	CHINA	ASIA
CAIRO	EGYPT	AFRICA
TRIPOLI	LIBYA	AFRICA
BEIJING	CHINA	ASIA
ROME	ITALY	EUROPE
TOKYO	JAPAN	ASIA
SYDNEY	AUSTRALIA	AUSTRALIA
SPARTA	GREECE	EUROPE
MADRID	SPAIN	EUROPE

3.37 写出 SQL 查询，结果表可列出城市名以字母 S 开始的所有城市及其所在国家。

3.38 写出 SQL 查询，结果表可列出以字母 O 结束的所有城市及其所在国家。

3.39 写出 SQL 查询，结果表可列出以字母 M 开始并正好含六个字母的所有城市及其所在国家。

3.40 写出 SQL 查询，结果表可列出第二个字母为 A 的所有城市及其所在国家。

3.41 写出 SQL 查询，结果表可列出既不在北美，也不在南美，且名称不为六个字母的所有城市及其所在国家。

3.42 写出SQL查询，结果表可列出在非洲、澳大利亚和亚洲的所有城市、其所在国家及洲。

补充题答案

3.23

```
SELECT id, date_ordered, sales_rep_id, total
FROM s_ord
WHERE sales_rep_id = '14'
AND date_ordered = '31-AUG-92';
ID  DATE_ORDE SAL  TOTAL
--- --------- --- ------
101 31-AUG-92 14  8056.6
98  31-AUG-92 14     595
99  31-AUG-92 14    7707
```

3.24

```
SELECT id, date_ordered, sales_rep_id, total
FROM s_ord
WHERE sales_rep_id = '14'
AND date_ordered = '31-AUG-92'
AND total > 6000;
ID  DATE_ORDE SAL  TOTAL
--- --------- --- ------
101 31-AUG-92 14  8056.6
99  31-AUG-92 14    7707
```

3.25

```
SELECT id, date_ordered, sales_rep_id, payment_type
FROM s_ord
WHERE date_ordered = '31-AUG-92'
OR payment_type = 'CASH';
ID  DATE_ORDE SAL PAYMEN
--- --------- --- ------
100 31-AUG-92 11  CREDIT
101 31-AUG-92 14  CREDIT
103 02-SEP-92 15  CASH
110 09-SEP-92 11  CASH
111 09-SEP-92 11  CASH
98  31-AUG-92 14  CASH
99  31-AUG-92 14  CREDIT
112 31-AUG-92 12  CREDIT
```

3.26

```
SELECT id, date_ordered, sales_rep_id, payment_type
FROM s_ord
WHERE NOT date_ordered = '31-AUG-92';
ID  DATE_ORDE SAL PAYMEN
--- --------- --- ------
102 01-SEP-92 15  CREDIT
103 02-SEP-92 15  CASH
104 03-SEP-92 15  CREDIT
105 04-SEP-92 11  CREDIT
106 07-SEP-92 12  CREDIT
107 07-SEP-92 15  CREDIT
108 07-SEP-92 13  CREDIT
109 08-SEP-92 11  CREDIT
110 09-SEP-92 11  CASH
111 09-SEP-92 11  CASH
97  28-AUG-92 12  CREDIT
```

3.27

```
SELECT id, date_ordered, sales_rep_id, payment_type
FROM s_ord
WHERE NOT date_ordered = '31-AUG-92'
AND payment_type = 'CREDIT';
ID  DATE_ORDE SAL PAYMEN
--- --------- --- ------
102 01-SEP-92 15  CREDIT
104 03-SEP-92 15  CREDIT
105 04-SEP-92 11  CREDIT
106 07-SEP-92 12  CREDIT
107 07-SEP-92 15  CREDIT
108 07-SEP-92 13  CREDIT
109 08-SEP-92 11  CREDIT
97  28-AUG-92 12  CREDIT
```

3.28

```
SELECT id, date_ordered, sales_rep_id, total
FROM s_ord
WHERE sales_rep_id = '12'
AND (date_ordered = '31-AUG-92' OR date_ordered = '01-SEP-92');
ID  DATE_ORDE SAL TOTAL
--- --------- --- ----------
112 31-AUG-92 12  550
```

3.29

```
SELECT id, date_ordered, date_shipped
FROM s_ord
WHERE date_ordered <= '31-AUG-92'
OR date_shipped <= '10-SEP-92';
ID  DATE_ORDE DATE_SHIP
--- --------- ---------
100 31-AUG-92 10-SEP-92
101 31-AUG-92 15-SEP-92
102 01-SEP-92 08-SEP-92
108 07-SEP-92 10-SEP-92
97  28-AUG-92 17-SEP-92
98  31-AUG-92 10-SEP-92
99  31-AUG-92 18-SEP-92
112 31-AUG-92 10-SEP-92
```

3.30

```
SELECT id, date_shipped, payment_type
FROM s_ord
WHERE NOT (date_shipped > '15-SEP-92'
AND payment_type = 'CREDIT')
ORDER BY date_shipped;
ID  DATE_SHIP PAYMEN
--- --------- ------
102 08-SEP-92 CREDIT
100 10-SEP-92 CREDIT
108 10-SEP-92 CREDIT
112 10-SEP-92 CREDIT
98  10-SEP-92 CASH
101 15-SEP-92 CREDIT
106 15-SEP-92 CREDIT
110 21-SEP-92 CASH
111 21-SEP-92 CASH
103 22-SEP-92 CASH
```

3.31

```
SELECT id, date_ordered, date_shipped, total
FROM s_ord
WHERE total > 6000
AND (NOT date_ordered > '31-AUG-92'
OR NOT date_shipped < '10-SEP-92');
ID  DATE_ORDE DATE_SHIP    TOTAL
--- --------- --------- -------
100 31-AUG-92 10-SEP-92  601100
```

```
101 31-AUG-92 15-SEP-92   8056.6
104 03-SEP-92 23-SEP-92    32430
106 07-SEP-92 15-SEP-92    15634
107 07-SEP-92 21-SEP-92   142171
108 07-SEP-92 10-SEP-92   149570
109 08-SEP-92 28-SEP-92  1020935
97  28-AUG-92 17-SEP-92    84000
99  31-AUG-92 18-SEP-92     7707
```

3.32

```
SELECT id, date_ordered, date_shipped, total
FROM s_ord
WHERE date_ordered LIKE '%AUG%';
```

由于日期列是固定长度，因此如下语句将产生相同结果：

```
SELECT id, date_ordered, date_shipped, total
FROM s_ord
WHERE date_ordered LIKE '___AUG___';
ID  DATE_ORDE DATE_SHIP    TOTAL
--- --------- ---------   ------
100 31-AUG-92 10-SEP-92   601100
101 31-AUG-92 15-SEP-92   8056.6
97  28-AUG-92 17-SEP-92    84000
98  31-AUG-92 10-SEP-92      595
99  31-AUG-92 18-SEP-92     7707
112 31-AUG-92 10-SEP-92      550
```

3.33

```
SELECT id, date_ordered, sales_rep_id, total
FROM s_ord
WHERE total BETWEEN 6000 AND 8000;
ID  DATE_ORDE SAL   TOTAL
--- --------- --- ------
99  31-AUG-92 14     7707
```

3.34

```
SELECT id, date_ordered, date_shipped
FROM s_ord
WHERE date_ordered BETWEEN '01-SEP-92' AND '30-SEP-92';
ID  DATE_ORDE DATE_SHIP
--- ---- ---- ---------
102 01-SEP-92 08-SEP-92
103 02-SEP-92 22-SEP-92
104 03-SEP-92 23-SEP-92
105 04-SEP-92 18-SEP-92
106 07-SEP-92 15-SEP-92
```

```
107 07-SEP-92 21-SEP-92
108 07-SEP-92 10-SEP-92
109 08-SEP-92 28-SEP-92
110 09-SEP-92 21-SEP-92
111 09-SEP-92 21-SEP-92
```

3.35

```
SELECT id, customer_id, date_ordered
FROM s_ord
WHERE customer_id IN (204, 206, 201, 202)
ORDER BY customer_id;
ID  CUS DATE_ORDE
--- --- ---------
97  201 28-AUG-92
98  202 31-AUG-92
100 204 31-AUG-92
111 204 09-SEP-92
102 206 01-SEP-92
```

3.36

```
SELECT DISTINCT s_customer.name, s_customer.id,
    s_ord.payment_type "ORDERS"
FROM s_customer, s_ord
WHERE s_customer.id = s_ord.customer_id
ORDER BY s_customer.id;
NAME                 ID  ORDERS
-------------------- --- ------
One Sport            201 CREDIT
Deportivo Caracas    202 CASH
New Delhi Sports     203 CREDIT
Ladysport            204 CASH
Ladysport            204 CREDIT
Kim's Sporting Goods 205 CREDIT
Sportique            206 CREDIT
Muench Sports        208 CASH
Muench Sports        208 CREDIT
Beisbol Si!          209 CREDIT
Futbol Sonora        210 CREDIT
Helmut's Sports      211 CREDIT
Hamada Sport         212 CREDIT
Sports Emporium      213 CREDIT
Sports Retail        214 CASH
```

3.37

```
SELECT city, country
FROM world_cities
WHERE city LIKE 'S%';
```

```
CITY                      COUNTRY
------------------------- -------------------------
SANTIAGO                  CHILE
SHENYANG                  CHINA
SYDNEY                    AUSTRALIA
SPARTA                    GREECE
```

3.38

```
SELECT city, country
FROM world_cities
WHERE city LIKE '%O';
```

```
CITY                      COUNTRY
------------------------- -------------
TOLEDO                    UNITED STATES
RIO DE JANEIRO            BRAZIL
SANTIAGO                  CHILE
QUITO                     ECUADOR
CAIRO                     EGYPT
TOKYO                     JAPAN
```

3.39

```
SELECT city, country
FROM world_cities
WHERE city LIKE 'M_____';
```

```
CITY                      COUNTRY
------------------------- -------------------------
MADRAS                    INDIA
MOSCOW                    RUSSIA
MADRID                    SPAIN
```

3.40

```
SELECT city, country
FROM world_cities
WHERE city LIKE '_A%';
```

```
CITY                      COUNTRY
------------------------- -------------------------
DALLAS                    UNITED STATES
NASHVILLE                 UNITED STATES
VANCOUVER                 CANADA
WARSAW                    POLAND
SANTIAGO                  CHILE
CARACAS                   VENEZUELA
MADRAS                    INDIA
```

```
MANCHESTER                    ENGLAND
PARIS                         FRANCE
CAIRO                         EGYPT
MADRID                        SPAIN
```

3.41

```
SELECT city, country
FROM world_cities
WHERE city NOT LIKE '_____' AND
      continent NOT IN ('NORTH AMERICA', 'SOUTH AMERICA');
CITY                          COUNTRY
----------------------------- -------------------------
NEW DELHI                     INDIA
MANCHESTER                    ENGLAND
PARIS                         FRANCE
SHENYANG                      CHINA
CAIRO                         EGYPT
TRIPOLI                       LIBYA
BEIJING                       CHINA
ROME                          ITALY
TOKYO                         JAPAN
```

3.42

```
SELECT city, country, continent
FROM world_cities
WHERE continent IN ('ASIA', 'AFRICA', 'AUSTRALIA');
CITY                          COUNTRY                   CONTINENT
----------------------------- ------------------------- -----------
MADRAS                        INDIA                     ASIA
NEW DELHI                     INDIA                     ASIA
BOMBAY                        INDIA                     ASIA
SHENYANG                      CHINA                     ASIA
CAIRO                         EGYPT                     AFRICA
TRIPOLI                       LIBYA                     AFRICA
BEIJING                       CHINA                     ASIA
TOKYO                         JAPAN                     ASIA
SYDNEY                        AUSTRALIA                 AUSTRALIA
```

第4章　算术运算及内部函数

4.1　算术运算

在第1章中，我们讲到的SQL是用于检索信息的数据库语言。它虽然不是针对复杂的数学处理而设计的，但有时仍希望它能用表中的数值或字符列来进行简单的运算。运算结果产生的新列不能成为该表的永久字段，它们仅仅是为了显示之用。

SQL支持算术运算符，它们的优先级列在表4-1中。这个优先级沿用标准的算术运算顺序，即先乘除后加减。在大多数程序设计语言中，相同运算符的计算是从左到右。圆括号总是用来改变计算的顺序或仅仅是为了清楚起见。

表4-1　算术运算符的先后顺序

运算符	说明	优先级
( )	圆括号	最先计算
* /	乘、除	接下来计算
+ -	加、减	最后计算

计算常常用来跟踪存货。表4-2给出了一个存货表S_INVENTORY的子集，在下面的各节中将会用到它。

表4-2　存货表

PRODUCT	WAREHOU	AMOUNT_IN_STOCK	REORDER_POINT	MAX_IN_STOCK	RESTOCK_D
20106	101	993	625	1000	
20108	101	700	700	1225	
20510	101	1389	850	1400	
20512	101	850	850	1450	
30421	101	1822	1800	3150	
40422	101	0	350	600	08-FEB-93
20106	201	220	150	260	
20108	201	166	150	260	
20510	201	175	100	175	
20512	201	162	100	175	
30421	201	102	80	140	
30433	201	130	130	230	

下面的例子介绍了在SQL查询中算术运算的使用。

例4.1

Warehouse 101的仓库经理不想盘点存货，认为对每种存货再订购100件是一件较为容易的事。写出SQL查询，显示目前哪些产品有存货。如果Warehouse 101对每种存货再订购100件的话，存货会有多少？

```
SELECT product_id, amount_in_stock, amount_in_stock+100
FROM s_inventory
WHERE warehouse_id='101';
PRODUCT AMOUNT_IN_STOCK AMOUNT_IN_STOCK+100
------- --------------- -------------------
20106               993                1093
20108               700                 800
20510              1389                1489
20512               850                 950
30421              1822                1922
40422                 0                 100
```

注意，在结果表中，新列被命名为AMOUNT_IN_STOCK+100，该命名虽然不是很有创意性，但具有描述性。记住，这个新列在表中并不存在，它是一个伪列。在结果表中，可通过定义别名给该伪列一个新名。

例4.2

仓库经理不想让别人注意到额外的存货，他想知道为每项产品订购100件后，是否会高于管理所规定的最大值。写出一个SQL查询，回答他的问题。

```
SELECT product_id, max_in_stock-(amount_in_stock+100) "RESULT"
FROM s_inventory
WHERE warehouse_id = '101';
PRODUCT    RESULT
------- ---------
20106          -93
20108          425
20510          -89
20512          500
30421         1228
40422          500
```

注意，在这个计算中用到了两个列名（max_in_stock及amount_in_stock）。实际计算中可使用任意多的列。经理可以从结果表中看出，如果进行此次订购的话，有两项（20106和20510）会超过存货规定的最大值。

例 4.3

Warehouse 201 的仓库经理预测将会有人订购 20106 及 20108 产品，她担心这些产品的存货会减少一半。如果她的担心是正确的话，写出一个 SQL 查询，显示这两项还会剩下多少存货。

```
SELECT product_id, amount_in_stock, amount_in_stock /2
FROM s_inventory
WHERE warehouse_id = '201'
AND (product_id = '20106' OR product_id = '20108');
PRODUCT AMOUNT_IN_STOCK AMOUNT_IN_STOCK/2
------- --------------- -----------------
20106               220               110
20108               166                83
```

例 4.4

例 4.3 给出了这些产品的存货量，该经理需要知道，这些存货量是否低于再订购点以及低多少。写出回答这个问题的 SQL 查询，将新列命名为“ORDER”。

```
SELECT product_id, amount_in_stock, reorder_point,
     reorder_point - (amount_in_stock/2) "ORDER"
FROM s_inventory
WHERE warehouse_id = '201' AND
(product_id = '20106' OR product_id = '20108');
PRODUCT AMOUNT_IN_STOCK REORDER_POINT      ORDER
------- --------------- ------------- ---------
20106               220           150        40
20108               166           150        67
```

现在经理知道她该订购 20106 产品 40 件；20108 产品 67 件。在此例中，为了清楚起见而使用了圆括号，但它是可以省略的。因为省略它不影响运算的先后顺序。而在下面给出的例子中，圆括号对产生正确结果是必须的。

例 4.5

Warehouse 201 的仓库经理决定对 20106 产品及 20108 产品各订购 100 件，产品到货后，如果所得到的每个产品新数量的一半被卖掉，还会剩多少件？它们仍然高于再订购点吗？写一个 SQL 查询，回答这个问题。

```
SELECT product_id, amount_in_stock, (amount_in_stock + 100)/2,
     reorder_point
FROM s_inventory
WHERE warehouse_id = '201' AND
(product_id = '20106' OR product_id = '20108');
```

```
PRODUCT AMOUNT_IN_STOCK (AMOUNT_IN_STOCK+100)/2 REORDER_POINT
------- ---------------- ------------------------ --------------
20106                220                      160            150
20108                166                      133            150
```

如果该语句不用圆括号写出，就会是 amount in _ stock + 100/2，结果存货数仅增加了 50，所得值为 270 及 216。为了明确起见，不管圆括号对正确的计算是否必须，在任何情况下都使用圆括号，这不失为一个好主意。

4.2 内部函数

SQL 语言为操纵数据提供了许多内部函数，可分成不同的种类。表 4-3 给出了它们的类型以及在本书中对其进行讲解的章节。

表 4-3 内部函数的类型

操作对象	相关章节
单一行中的单独的数值	4.3 节
单一行中的单独的字符值	4.4 节
完整的列或行的组成数据	第 5 章
日期和时间值	第 6 章

图 4-1 讲解了 SQL 函数的处理过程。每个函数有一个名称（通常是对任务的描述）并能接受一个或多个自变量。自变量往往是一个列名，但有时也可能是一个变量值、一个常数或一个表达式。某些函数的输出结果确实改变了每一列中的值，而另一些函数的输出结果则是对有关信息进行报告。

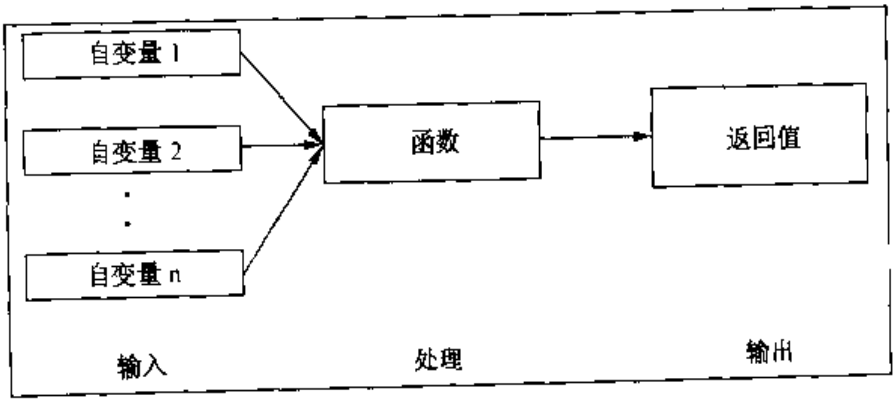


图 4-1 SQL 函数处理

调用内部函数的语法以下：

function _ name (column or expression [, argument2, …])

自变量的个数取决于函数。第一个自变量往往是一个列名或者使用列名进行计算的表达式。

4.3　数值函数

本节描述一些通用函数，它们对单独的数值进行运算。表 4-4 列出了一些最常用的函数，并对每一个函数进行了说明。其他用于科学计算的函数（如正弦、余弦、正切函数等）在此处未给出，但在需要时也可采用。

表 4-4　数值函数

函数名	输入参数	返回值
ABS(m)	m = 值	m 的绝对值
MOD(m,n)	m = 值，n = 除数	m 被 n 除后的余数
POWER(m,n)	m = 值，n = 幂	m 的 n 次方
ROUND(m[,n])	m = 值，n = 小数位（可选）	m 四舍五入至小数点后 n 位的值（n 缺省为 0）
TRUNC(m[,n])	m = 值，n = 小数位（可选）	m 截断至 n 位小数位的值（n 缺省为 0）

通常数值函数在一个特定列中操纵数据，它们对每一行起作用，并对每一行返回一个结果，该结果可以是一个插入到该列的新值，也可以是关于这列中值的信息。

4.3.1　ABS(m)函数

一个数的绝对值等于忽略了正、负号的数值。当值高于或低于指标并不重要的时候，通常用该函数发现数与数之间的差异。

例 4.6

对每项存货，写出一个 SQL 查询，显示存货量与再订购点之间的差值以及该差值的绝对值。

```
SELECT product_id, warehouse_id,
       (amount_in_stock - reorder_point) "DIFFERENCE",
       ABS(amount_in_stock - reorder_point)
FROM s_inventory;
PRODUCT WAREHOU DIFFERENCE ABS(AMOUNT_IN_STOCK-REORDER_POINT)
------- ------- ---------- ------------------------------------
20106   101            368                                  368
20108   101              0                                    0
20510   101            539                                  539
20512   101              0                                    0
30421   101             22                                   22
40422   101           -350                                  350
20106   201             70                                   70
20108   201             16                                   16
20510   201             75                                   75
20512   201             62                                   62
30421   201             22                                   22
30433   201              0                                    0
```

ABS( ) 函数返回的值不能指出存货数是高于还是低于再订购点，只能表明它们之间相差多少。注意，列标题是包括函数名的整个表达式。通常在结果表中，通过使用别名来使用一个更具有描述性的列名较好。例如，“DIFFERENCE”列。

4.3.2 MOD(m, n)函数

该函数给出 m 被 n 除后的余数。例如，函数 MOD (26, 3)，返回值为 2，如下所示。

```
    8 ←— 商
3)26
  24
  --
   2 ←— 余数
```

记住，m 称为被除数，n 称为除数 。

例 4.7

写出一个 SQL 查询，显示每项存货量被 9 除后的余数。

```
SELECT product_id, warehouse_id, amount_in_stock,
MOD(amount_in_stock, 9)
FROM s_inventory;
```

PRODUCT	WAREHOU	AMOUNT_IN_STOCK	MOD(AMOUNT_IN_STOCK,9)
20106	101	993	3
20108	101	700	7
20510	101	1389	3
20512	101	850	4
30421	101	1822	4
40422	101	0	0
20106	201	220	4
20108	201	166	4
20510	201	175	4
20512	201	162	0
30421	201	102	3
30433	201	130	4

4.3.3 POWER(m,n)函数

POWER (m,n) 函数返回第一个自变量以第二个自变量作为幂的结果，即 m 的 n 次幂。第二个自变量必须是一个整数。

例 4.8

写出一个 SQL 查询，显示每项存货量及存货量的平方。

```
SELECT amount_in_stock, POWER(amount_in_stock, 2)
FROM s_inventory;
AMOUNT_IN_STOCK  POWER(AMOUNT_IN_STOCK,2)
---------------- ------------------------
            993                   986049
            700                   490000
           1389                  1929321
            850                   722500
           1822                  3319684
              0                        0
            220                    48400
            166                    27556
            175                    30625
            162                    26244
            102                    10404
            130                    16900
```

4.3.4 ROUND(m[,n])函数

ROUND(m[,n]) 函数用于计算 m 四舍五入到小数点后 n 位的值。如果没有指出小数位的自变量，则缺省为 0，即四舍五入到最接近整数的值。也可以用 -1 或 -2 分别表示四舍五入到邻近的十位或百位数。下面的三个例子对此进行了说明。

例 4.9

```
将每项存货量用9除，将结果显示为最接近整数的四舍五入值。
SELECT amount_in_stock/9, ROUND(amount_in_stock/9)
FROM s_inventory;
AMOUNT_IN_STOCK/9 ROUND(AMOUNT_IN_STOCK/9)
----------------- -----------------
        110.33333               110
         77.777778                78
        154.33333               154
        94.444444                94
        202.44444               202
                0                 0
        24.444444                24
        18.444444                18
        19.444444                19
               18                18
        11.333333                11
        14.444444                14
```

注意，由于没有指出小数位，故缺省值为0，即四舍五入到最接近整数的值。该函数也可以写为ROUND(amount_in_stock/9, 0)，所得结果是一样的。

例4.10

将每项存货量用9除，显示四舍五入到小数点后一位和后两位的结果。

```
SELECT amount_in_stock/9, ROUND(amount_in_stock/9,1) "TO TENTH",
     ROUND(amount_in_stock/9,2) "TO HUNDREDTH"
FROM s_inventory;
AMOUNT_IN_STOCK/9  TO TENTH  TO HUNDREDTH
----------------- --------- ------------
        110.33333     110.3        110.33
        77.777778      77.8         77.78
        154.33333     154.3        154.33
        94.444444      94.4         94.44
        202.44444     202.4        202.44
                0         0             0
        24.444444      24.4         24.44
        18.444444      18.4         18.44
        19.444444      19.4         19.44
               18        18            18
        11.333333      11.3         11.33
        14.444444      14.4         14.44
```

例4.11

将每项存货量用9除，显示结果到最邻近的十位及百位数。

```
SELECT amount_in_stock/9, ROUND(amount_in_stock/9,-1) "TO TENS",
     ROUND(amount_in_stock/9,-2) "TO HUNDREDS"
FROM s_inventory;
AMOUNT_IN_STOCK/9  TO TENS  TO HUNDREDS
----------------- -------- -----------
        110.33333      110         100
        77.777778       80         100
        154.33333      150         200
        94.444444       90         100
        202.44444      200         200
                0        0           0
        24.444444       20           0
        18.444444       20           0
        19.444444       20           0
               18       20           0
        11.333333       10           0
        14.444444       10           0
```

注意，当值20四舍五入到最邻近的百位数时，结果为0；同样道理，值51四舍五入到最邻近的百位数时，即为100。

对负数的四舍五入也是同样的道理。内部表DUAL用于阐述对负数的四舍五入。

例4.12

有两个数：-55.55及-44.44，显示ROUND()函数如何将这些数四舍五入至邻近的十位数、整数及十分位数的。使用两个SECECT语句。

```
SELECT ROUND(-55.55, -1), ROUND(-55.55, 0), ROUND(-55.55, 1)
FROM DUAL;
ROUND(-55.55,-1) ROUND(-55.55,0) ROUND(-55.55,1)
-- ------------ -------------- ---------------
             -60            -56           -55.6

SELECT ROUND(-44.44, -1), ROUND(-44.44, 0), ROUND(-44.44, 1)
FROM DUAL;
ROUND(-44.44,-1) ROUND(-44.44,0) ROUND(-44.44,1)
---------------- -------------- ---------------
             -40            -44           -44.4
```

4.3.5 TRUNC(m[,n])函数

TRUNC(m[,n])函数用于计算将m在n位截断的值。如果TRUNC(m[,n])函数没有n的自变量，则缺省为0，所得的截断值为小于或等于m的最邻近的整数。所有的小数值均被忽略。例如99.99被截断为99，使用-1或-2时，该值可被截断至最邻近的十位或百位数。举例如下。

例4.13

将每项存货量用9除，将结果截断显示为最邻近的整数值。

```
SELECT amount_in_stock/9, TRUNC(amount_in_stock/9)
FROM s_inventory;
AMOUNT_IN_STOCK/9 TRUNC(AMOUNT_IN_STOCK/9)
----------------- ------------------------
        110.33333                      110
        77.777778                       77
        154.33333                      154
        94.444444                       94
        202.44444                      202
                0                        0
        24.444444                       24
        18.444444                       18
        19.444444                       19
               18                       18
        11.333333                       11
        14.444444                       14
```

将这些结果与例 4.9 的结果进行比较。注意，超过 0.5 的十进制值被四舍五入。如 77.777778 被四舍五入为 78，而截断为 77。对小于 0.5 的十进制值，如 110.33333，四舍五入值及截断值是一样的。

例 4.14

将每项存货量用 9 除，将结果截断显示为最邻近的 1/10 及 1/100 的值。

```
SELECT amount_in_stock/9, TRUNC(amount_in_stock/9,1) "TO TENTH",
    TRUNC(amount_in_stock/9,2) "TO HUNDREDTH"
FROM s_inventory;
AMOUNT_IN_STOCK/9   TO TENTH    TO HUNDREDTH
----------------- ---------  ------------
        110.33333      110.3         110.33
        77.777778       77.7          77.77
        154.33333      154.3         154.33
        94.444444       94.4          94.44
        202.44444      202.4         202.44
                0          0              0
        24.444444       24.4          24.44
        18.444444       18.4          18.44
        19.444444       19.4          19.44
               18         18             18
        11.333333       11.3          11.33
        14.444444       14.4          14.44
```

例 4.15

将每项存货量用 9 除，这次将结果截断显示为邻近的十位及百位数。

```
SELECT amount_in_stock/9, TRUNC(amount_in_stock/9,-1) "TO TENS",
     TRUNC(amount_in_stock/9,-2) "TO HUNDREDS"
FROM s_inventory;
AMOUNT_IN_STOCK/9    TO TENS   TO HUNDREDS
---- ------------ ---------  -----------
        110.33333        110          100
        77.777778         70            0
        154.33333        150          100
        94.444444         90            0
        202.44444        200          200
                0          0            0
        24.444444         20            0
        18.444444         10            0
        19.444444         10            0
               18         10            0
        11.333333         10            0
        14.444444         10            0
```

再一次将这些结果与例 4.11 的结果进行比较，可以看出四舍五入值及截断值的差异。想要获得什么样的结果，选择正确的函数是至关重要的。

4.3.6　嵌套函数

嵌套函数意味着使用一个函数的结果作为另一个函数的自变量。内部函数可在同一表达式中嵌套，有时这种嵌套对获得想要的结果是关键的。

例 4.16

写出一个 SQL 查询，显示每次存货数被 81 除后的余数四舍五入到最邻近的十位数的值。

```
SELECT amount_in_stock, MOD(amount_in_stock,81) "MOD 81",
    ROUND(MOD(amount_in_stock,81),-1) "ROUNDED"
FROM s_inventory;
```

AMOUNT_IN_STOCK	MOD 81	ROUNDED
993	21	20
700	52	50
1389	12	10
850	40	40
1822	40	40
0	0	0
220	58	60
166	4	0
175	13	10
162	0	0
102	21	20
130	49	50

在此例中，MOD( ) 函数的结果变成了 ROUND( ) 的第一个自变量。如果将语句倒过来写：

```
MOD (ROUND (amount  in_ stock, 81), -1) "ROUNDED"
```

结果就完全不一样，这样做会得到一个错误结果。当用嵌套函数时，必须仔细决定应首先计算哪个函数。

4.4　字符函数

对由一个或多个字符组成的单一值的运算，可采用若干通用的函数。这些值往往称为串。表 4-5 列出了一些最常用的函数，下面将一一进行解释。如果需要的话，也可采用其他一些函数。对字符串运算的函数可以转换一个或多个字母的大小写，用某种方法操纵字符串

或者返回关于串的信息。

表 4-5 字符函数

大小写转换		
函数名	输入参数	返回值
INITCAP(st)	st = 字符串值	返回 st 将每个单词的首字母大写，所有其他字母小写
LOWER(st)	st = 字符串值	返回 st 将每个单词的字母全部小写
UPPER(st)	st = 字符串值	返回 st 将每个单词的字母全部大写
字符操作		
函数名	**输入参数**	**返回值**
CONCAT(st1, st2)	st1 = 字符串值 st2 = 字符串值	返回 st 为 st2 接 st1 的末尾（可用操作符 "‖"）
LPAD(st1, n [, st2])	st1 = 字符串值 n = 整数值 st2 = 字符串值	返回右对齐的 st，st 为在 st1 的左边用 st2 填充直至长度为 n，st2 的缺省为空格
RPAD(st1, n [, st2])	st1 = 字符串值 n = 整数值 st2 = 字符串值	返回左对齐的 st，st 为在 st1 的右边用 st2 填充直至长度为 n，st2 的缺省为空格
LTRIM(st [, set])	st = 字符串值 set = 字符组	返回 st，st 为从左边删除 set 中字符直到第一个不是 set 中的字符，缺省时，指的是空格
RTRIM(st [, set])	st = 字符串值 set = 字符组	返回 st，st 为从右边删除 set 中字符直到第一个不是 set 中的字符，缺省时，指的是空格
REPLACE(st, search_st [, replace_st])	st = 字符串值 search_st = 目标串 replace_st = 替换串	将每次在 st 中出现的 search_st 用 replace_st 替换，返回一个 st，缺省时，删除 search_st
SUBSTR(st, m [, n])	st = 字符串值 m = 起始位置 n = 字符数	n = 返回 st 串的子串，从 m 位置开始，取 n 个字符长。缺省时，一直返回到 st 末端
返回信息		
函数名	**输入参数**	**返回值**
LENGTH(st)	st = 字符串值	数值，返回 st 中的字符数。
INSTR(st1, st2 [, m [, n]])	st1 = 字符串值 st2 = 字符串值 m = 起始位置 n = 在 st1 中，st2 出现的次数	数值，返回 st1 从第 m 字符开始，st2 第 n 次出现的位置，m 及 n 的缺省值为 1

表 4-6 示出了下列各节要使用的客户表 S_CUSTOMER 的子集。

表 4-6 S_CUSTOMER 的子集

NAME	ADDRESS	CITY	STATE
Sports,Inc	72 High St	Harrisonburg	VA
Toms Sporting Goods	6741 Main St	Harrisonburg	VA
Athletic Attire	54 Market St	Harrisonburg	VA
Athletics For All	286 Main St	Harrisonburg	VA
Shoes for Sports	538 High St	Harrisonburg	VA
BJ Athletics	632 Water St	Harrisonburg	VA
Athletics One	912 Columbia Rd	Lancaster	PA
Great Athletes	121 Litiz Pike	Lancaster	PA
Athletics Two	435 High Rd	Lancaster	PA
Athletes Attic	101 Greenfield Rd	Lancaster	PA

4.4.1 大小写转换函数

当不知道大小写时，第一组函数对数据库的检索是必须的。通常人们键入的数据并不全都是大写或全都是小写字母，有些列可能包含大小写混合的数据。这些函数可通过 SELECT 语句的使用，将数据按所希望的方式格式化。

INITCAP(st) 函数是将获得的一个字符串值转换成为首字符大写、其余小写的字符串，但该字符串自身在数据库内并未改变。

例 4.17

如果一个名字串为“THOMAS”、“thomas”或任意其他混合的大小写，使用这个函数会返回“Thomas”。使用内部表 DUAL 演示该 SQL 查询。

```
SELECT INITCAP('THOMAS'), INITCAP('thomas')
FROM DUAL;
INITCA INITCA
------ ------
Thomas Thomas
```

UPPER(st) 及 LOWER(st) 可用于检索字符串数据，并将数据全以大写或全以小写字母显示。

例 4.18

写出将 S_CUSTOMER 表中顾客名以下列三种方式显示的 SQL 查询：(1) 如表中所述的方式；(2) 全以大写字母表示；(3) 全以小写字母表示

```
SELECT name, UPPER(name), LOWER(name)
FROM s_customer;
NAME                 UPPER(NAME)          LOWER(NAME)
-------------------- -------------------- --------------------
Sports,Inc           SPORTS,INC           sports,inc
Toms Sporting Goods  TOMS SPORTING GOODS  toms sporting goods
Athletic Attire      ATHLETIC ATTIRE      athletic attire
Athletics For All    ATHLETICS FOR ALL    athletics for all
Shoes for Sports     SHOES FOR SPORTS     shoes for sports
BJ Athletics         BJ ATHLETICS         bj athletics
Athletics One        ATHLETICS ONE        athletics one
Great Athletes       GREAT ATHLETES       great athletes
Athletics Two        ATHLETICS TWO        athletics two
Athletes Attic       ATHLETES ATTIC       athletes attic
```

4.4.2 并置函数

字符串能通过 CONCAT(st1, st2) 函数进行组合。该函数返回包含两个串的一个新串。CONCAT(st1, st2) 只能将两个串连接在一起。

例 4.19

使用内部表 DUAL，将下列名字和姓氏并置：'Thomas'、'Jefferson'、'Benjamin' 及 'Franklin'。

```
SELECT CONCAT('Thomas','Jefferson') "FIRST",
       CONCAT('Benjamin','Franklin') "SECOND"
FROM DUAL;
FIRST              SECOND
------------------ ----------------
ThomasJefferson BenjaminFranklin
```

注意，在两个单词之间没有空格。新串只是将两个串准确地连在了一起。如果需要一个空格，空格必须出现在一个串中，而在表中不可能出现这种情况。运用并置运算符（||）可将想要的置于单括号内的串连接起来。在需要并置一系列的字符串时，上面的例子可以用下述方式写出：

```
SELECT 'Thomas' || ' ' || 'Jefferson' "FIRST",
       'Benjamin' || ' ' || 'Franklin' "SECOND"
FROM DUAL;
FIRST              SECOND
------------------ ----------------
Thomas Jefferson Benjamin Franklin
```

例 4.20

写出一个SQL查询，将客户名打印在一列，将它们所在城市及州名打印在另一列

```
SELECT name, city || ', ' || state
FROM s_customer;
NAME                 CITY||','||STATE
-------------------- ------------------
Sports, Inc          Harrisonburg, VA
Toms Sporting Goods  Harrisonburg, VA
Athletic Attire      Harrisonburg, VA
Athletics For All    Harrisonburg, VA
Shoes for Sports     Harrisonburg, VA
BJ Athletics         Harrisonburg, VA
Athletics One        Lancaster, PA
Great Athletes       Lancaster, PA
Athletics Two        Lancaster, PA
Athletes Attic       Lancaster, PA
```

并置运算符也可用于将输出结果格式化为希望的风格，甚至是完整的句子。将每个串置于单引号内，但列名不能用单引号。

例 4.21

写出一个SQL查询，将S　CUSTOMER表的每一行打印为下面的句子：

```
The store <name> is located in <city, state>
```

```
SELECT 'The store ' || name || ' is located in ' ||
     city || ', ' || state || '.'
FROM s_customer;
 'THESTORE'||NAME||'ISLOCATEDIN'||CITY||','||STATE||'.'
-------------------------------------------------------
The store Sports, Inc is located in Harrisonburg, VA.
The store Toms Sporting Goods is located in Harrisonburg, VA.
The store Athletic Attire is located in Harrisonburg, VA.
The store Athletics For All is located in Harrisonburg, VA.
The store Shoes for Sports is located in Harrisonburg, VA.
The store BJ Athletics is located in Harrisonburg, VA.
The store Athletics One is located in Lancaster, PA.
The store Great Athletes is located in Lancaster, PA.
The store Athletics Two is located in Lancaster, PA.
The store Athletes Attic is located in Lancaster, PA.
```

4.4.3 填充及整理函数

函数 LPAD(st1, n[, st2]) 和 RPAD(st1, n[, st2]) 提供了将一系列字符并置于原字符串的左边或右边的方法。函数的长度参数指出了返回串的准确长度，其中包括原字符串。若省略第二个字符串，则以空格填充。

例 4.22

写出一个 SQL 查询，用空格填充客户名的左边直到列宽为 25，并用星号填充右边直到列宽为 15。

```
SELECT LPAD(name,25), RPAD(name,15, '*')
FROM s_customer;
LPAD(NAME,25)                RPAD(NAME,15, *')
------------------------- ---------------
                Sports,Inc Sports,Inc*****
      Toms Sporting Goods  Toms Sporting G
          Athletic Attire  Athletic Attire
     Athletics For All     Athletics For A
      Shoes for Sports     Shoes for Sport
          BJ Athletics     BJ Athletics***
        Athletics One      Athletics One**
       Great Athletes      Great Athletes*
        Athletics Two      Athletics Two**
       Athletes Attic      Athletes Attic*
```

注意，当第三个字符串省略时，字符串用空格填充。结果表也表明了如果指定了串宽度太小，字符串将被从右至左截短。填充的串可多于一个字符。如下所示。

例 4.23

写出一个 SQL 查询，在第一列显示客户名，以一系列的“…”及一个空格，填充至客户名右边，直至列宽为 30；在第二列显示信誉等级。

```
SELECT RPAD(name, 30, '... '), credit_rating
FROM s_customer;
RPAD(NAME,30,'...') CREDIT_RA
------------------------------ ---------
Sports,Inc... ... ... ... ...  EXCELLENT
Toms Sporting Goods... ... ...  POOR
Athletic Attire... ... ... ...  GOOD
Athletics For All... ... ... ...  EXCELLENT
Shoes for Sports... ... ... ...  EXCELLENT
BJ Athletics... ... ... ... ...  POOR
```

```
Athletics One. . . . . .  GOOD
Great Athletes. . . . . .  EXCELLENT
Athletics Two. . . . . .  EXCELLENT
Athletes Attic. . . . . .  POOR
```

整理函数 RTRIM(st [, set]) 及 LTRIM(st [, set])，从串末尾去掉任意不想要的字符。这些函数的自变量是将被整理的列名或串以及要删除的字符集。如果未包含字符集时，将会去掉所有空格。键入数据的人故意在任意串前或者后加上空格是十分危险的。任意期望的字符集都可被同样的 LTRIM() 或 RTRIM() 函数去掉。整理函数在查找一个字符串实际长度时非常有用。这一点将会在 4.4.6 节讲到

例 4.24

写出一个 SQL 查询，去掉客户表中每一个地址末端的点及单词 St 和 Rd。

```
SELECT name, RTRIM(address,'. St Rd')
FROM s_customer;
NAME                    RTRIM(ADDRESS,'.STRD
----------------------- --------------------
Sports,Inc              72 High
Toms Sporting Goods     6741 Main
Athletic Attire         54 Marke
Athletics For All       286 Main
Shoes for Sports        538 High
BJ Athletics            632 Water
Athletics One           912 Columbia
Great Athletes          121 Litiz Pike
Athletics Two           435 High
Athletes Attic          101 Greenfiel
```

仔细检查结果表，可发现不仅 St 和 Rd 被去掉，而且在街名末端出现的任一 s、t、r 或 d 均被去掉（即列出的是 Marke，而不是 Market）。SQL 将字符集中的多个字符分别看成单个字符，而不是按单词处理。因此用整理函数时要小心，指定你想去掉的确切字符。

4.4.4　替换函数

替换函数 REPLACE(st, search_st[, replace_st]) 类似于大多数字处理程序中的“查找和替换”。在给出的每个实例中，目标字符或字符串被替换字符或字符串所代替。需要强调的是，使用这个函数时一定要小心。

例 4.25

写出一个SQL查询，将客户名称中的每一个字母'a'用星号替换。

a. 对该查询的第一次尝试如下：

```
SELECT name, REPLACE(name,'a','*')
FROM s_customer;
NAME                 REPLACE(NAME,'A','*'
-------------------- --------------------
Sports,Inc           Sports,Inc
Toms Sporting Goods  Toms Sporting Goods
Athletic Attire      Athletic Attire
Athletics For All    Athletics For All
Shoes for Sports     Shoes for Sports
BJ Athletics         BJ Athletics
Athletics One        Athletics One
Great Athletes       Gre*t Athletes
Athletics Two        Athletics Two
Athletes Attic       Athletes Attic
```

检查结果表后发现，只有小写的a被替换，因为在一个字符串中的字符是区分大小写的。为了克服这个问题，可使用大小写转换函数，即用UPPER('A')或用LOWER('a')函数。

b. 对该查询的第二次尝试较接近于想要的结果。

```
SELECT name, REPLACE(LOWER(name),'a','*')
FROM s_customer;
NAME                 REPLACE(LOWER(NAME),
-------------------- --------------------
Sports,Inc           sports,inc
Toms Sporting Goods  toms sporting goods
Athletic Attire      *thletic *ttire
Athletics For All    *thletics for *ll
Shoes for Sports     shoes for sports
BJ Athletics         bj *thletics
Athletics One        *thletics one
Great Athletes       gre*t *thletes
Athletics Two        *thletics two
Athletes Attic       *thletes *ttic
```

无论大写还是小写的a均被去掉，但是顾客名中的所有其他字母均被显示成小写字母。

c. 再试一次，将另一个函数INITCAP()与这个函数结合使用，以获得想要的结果。该查询的下一版本如下：

```
SELECT name, INITCAP(REPLACE(LOWER(name),'a','*'))
FROM s_customer;
NAME                 INITCAP(REPLACE(LOWE
-------------------- --------------------
Sports,Inc           Sports,Inc
Toms Sporting Goods  Toms Sporting Goods
Athletic Attire      *Thletic *Ttire
Athletics For All    *Thletics For *Ll
Shoes for Sports     Shoes For Sports
BJ Athletics         Bj *Thletics
Athletics One        *Thletics One
Great Athletes       Gre*T *Thletes
Athletics Two        *Thletics Two
Athletes Attic       *Thletes *Ttic
```

结果表仍然不能满足最初的要求。如果键入到一个表中的所有数据全为大写或全为小写，就不会出现这种问题。但这有时又是不可能的。因此，必须试一下多种查询方法，以找到一种最可接受的查询。还可以替换字符的序列。

例 4.26

写出一个SQL查询，将所有的'Athl'字符序列用'Asc'序列替换。该查询只对与这些字母具有同样大小写的序列起作用。

```
SELECT name, REPLACE(name, 'Athl','Asc')
FROM s_customer;
NAME                 REPLACE(NAME,'ATHL','ASC')
-------------------- ------------------------------
Sports,Inc           Sports,Inc
Toms Sporting Goods  Toms Sporting Goods
Athletic Attire      Ascetic Attire
Athletics For All    Ascetics For All
Shoes for Sports     Shoes for Sports
BJ Athletics         BJ Ascetics
Athletics One        Ascetics One
Great Athletes       Great Ascetes
Athletics Two        Ascetics Two
Athletes Attic       Ascetes Attic
```

4.4.5 子串函数

SUBSTR(st, m[, n]) 函数返回串自变量中从第 m 个字符位置开始，后接 n 个字符的字符串。如果长度 n 未给出，则该函数将返回直到串末端的所有字母。下面用内部表 DUAL

说明这个函数。

例 4.27

写出一个SQL查询，显示'ARCHIBALD BEARISOL'从第6个字符开始，后接9个字符的子串；显示从第11个字符开始，直到串末端的所有字母的子串；显示单词'ALEXANDER'的前4个字符；显示单词'ALEXANDER'的第5、6、7位的字符。

```
SELECT    SUBSTR('ARCHIBALD BEARISOL',6,9),
          SUBSTR('ARCHIBALD BEARISOL',11),
          SUBSTR ('ALEXANDER',1,4), SUBSTR ('ALEXANDER',5,3)
FROM DUAL;
SUBSTR('A  SUBSTR('   SUBS   SUB
---------  --------   ----   ---
BALD BEAR  BEARISOL   ALEX   AND
```

仔细检查结果表，可看出一个空格也代表一个字符，在计算子串时，需将空格计算在内。

4.4.6 字符长度函数

最后的两个字符函数均返回整数。再强调一次，一个字符串中的全部字符均被计算在内，包括空格、小数点、逗号等。LENGTH(st) 返回字符串的字符个数，INSTR(st1，st2[，m[，n]]) 函数返回字符串 st2 在字符串 st1 从 m 位置开始，第 n 次出现时所在的位置。如果 m 及 n 缺省的话，缺省值为 1

例 4.28

写出一个SQL查询，返回每个客户名的准确字符数。为了确保不包括名称后面的空格，使用 RTRIM() 函数删除之。

```
SELECT name, LENGTH(RTRIM(name))
FROM s_customer;
NAME                    LENGTH(RTRIM(NAME))
----------------------  -------------------
Sports,Inc                                10
Toms Sporting Goods                       19
Athletic Attire                           15
Athletics For All                         17
Shoes for Sports                          16
BJ Athletics                              12
Athletics One                             13
Great Athletes                            14
Athletics Two                             13
Athletes Attic                            14
```

例 4.29

许多客户名不只一个单词。对多于两个单词的客户，使用 INSTRA() 函数，分别返回每个客户名中第一个、第二个空格的位置。第一次尝试查询如下：

```
SELECT name, INSTR(name,' '), INSTR(name,' ',1,2)
FROM s_customer;
NAME                 INSTR(NAME,' ') INSTR(NAME,' ',1,2)
-------------------- --------------- -------------------
Sports,Inc                         0                   0
Toms Sporting Goods                5                  14
Athletic Attire                    9                   0
Athletics For All                 10                  14
Shoes for Sports                   6                  10
BJ Athletics                       3                   0
Athletics One                     10                   0
Great Athletes                     6                   0
Athletics Two                     10                   0
Athletes Attic                     9                   0
```

仔细检查结果表，可发现一个有趣的事实：当键入第一个客户名称时，在逗号及 I 之间无空格。如果希望在二者之间有一个空格的话，该函数将帮助找出这个问题，然后可使用 UPDATE 语句更正（见第 1 章）。

例 4.30

假设有一个 PRESIDENT 表，名字列由姓氏、一个逗号、一个空格和名字组成，如 Jefferson, Thomas 及 Lincoln, Abraham。写出一个 SQL 查询，使名字列由名字、姓氏及中间一个空格组成，没有逗号，如 Thomas Jefferson。

该任务的完成需用多个函数，可按以下三步进行：

a. 找到逗号的位置。

b. 取出姓氏及名字。

c. 将姓氏并置于名字之后。

对每一步的说明如下：

a. 使用 INSTR() 函数找到逗号的位置，可写出下列查询。

```
SELECT name, INSTR(name, ',')
FROM president;
NAME                      INSTR(NAME,',')
------------------------- ---------------
Jefferson, Thomas                      10
Lincoln, Abraham                        8
```

b. 使用 LENGTH() 函数及 SUBSTR() 函数取出名字及姓氏。名字从逗号后的第二个位置开始，姓氏在逗号前的位置结束。

```
SELECT name, SUBSTR(name, INSTR(name, ',')+2) "First",
           SUBSTR(name, 1, INSTR(name, ',')-1) "Second"
FROM president;
NAME                      First                Second
------------------------- -------------------- ---------
Jefferson, Thomas         Thomas               Jefferson
Lincoln, Abraham          Abraham              Lincoln
```

c. 上述查询为名字及姓氏产生了两个串，现在要做的是将这两个串并置成一个字符串，中间有一个空格。

```
SELECT name, SUBSTR(name, INSTR(name, ',')+2) || ' ' ||
           SUBSTR(name, 1, INSTR(name, ',')-1) "Reverse"
FROM president;
NAME                      Reverse
------------------------- ------------------------------------
Jefferson, Thomas         Thomas Jefferson
Lincoln, Abraham          Abraham Lincoln
```

4.5 重要的转换函数

内部函数是专门用于对数值或字符数据进行运算的，有时可能会引起一些问题。有时某些列可能包含 NULL 值，它使这些函数的使用复杂化。我们有时想对包含字符串数据的列使用数值函数，或者对包含数值数据的列使用字符函数。在使用该函数之前，数据必须已经被转换。表 4-7 列出了一些可用于操纵与转换数值及字符数据的常用函数，在需要时也可使用其他函数。

表 4-7 一些实用函数

函数名	输入参数	返回值
NVL(m, n)	m = 数或字符串值	如果 m 值为 NULL，返回值为 n，否则返回 m
TO_CHAR(m [, fmt])	m = 数值 fmt = 串格式	m 从一个数值转换为指定格式的字符串；fmt 缺省时，fmt 值的宽度正好能容纳所有的有效数字
TO_NUMBER(st [, fmt])	st = 字符串值 fmt = 数值格式	st 从字符型数据转换成按指定格式的数值，缺省时数值格式串的大小正好为整个数

4.5.1 NULL 值的运算及空值函数

回忆一下，在 SQL 中有一个特殊值 NULL，它或者是空值，或者没有任何值。NULL 代表一个未知的值，它与零不一样。由于零不会自动取代 NULL，因此对一个未知值或

NULL进行的任意运算，其结果为NULL。

例4.31

假设你有一个MULT表（如下所示），它包含两列数，其中某些值可能为NULL。写出SQL查询，显示两列数相乘的结果。

```
    FIRST     SECOND
--------- ----------
      987          2
       22
                  35
        5          4
```

```
SELECT first, second, first * second "RESULT"
FROM mult;
    FIRST     SECOND     RESULT
--------- ---------- ----------
      987          2       1974
       22
                  35
        5          4         20
```

注意结果列中的空白处。用一个未知数乘以别的数是不可能的。乘NULL值的结果会产生一个NULL值。我们通常不希望发生这样的事。NVL(m, n)可将NULL值转换成为了计算目的设定的一个缺省值。在必要时，NVL()函数也可用于字符串，返回值总是与函数值的数据类型一致。

例4.32

写出一个SQL查询，显示MULT表的两列数相乘的结果。此次用1替代NULL值。虽然也可用0替代，但它产生的结果总为0。

```
SELECT first, second, NVL(first, 1) * NVL(second, 1)
FROM mult;
    FIRST     SECOND NVL(FIRST,1)*NVL(SECOND,1)
--------- ---------- --------------------------
      987          2                       1974
       22                                    22
                  35                          35
        5          4                          20
```

例4.33

假设正存取一个名为NAMES的表（如下所示），该表中某些地方可能没有名字或姓氏。

写出一个SQL查询，将该表打印出来。如果在任一列中有NULL值，用单词'UNKNOWN'替代。

```
FIRSTNAME                      LASTNAME
------------------------------ ---------------
THOMAS                         JEFFERSON
                               SOCRATES

SELECT NVL(firstname, 'UNKNOWN'), NVL(lastname, 'UNKNOWN')
FROM names;
NVL(FIRSTNAME,'UNKNOWN')       NVL(LASTNAME,'UNKNOWN')
------------------------------ ------------------------------
THOMAS                         JEFFERSON
UNKNOWN                        SOCRATES
```

4.5.2 字符串与数值的转换

这里要解释的两个转换函数是TO_CHAR(m [, fmt]) 和TO_NUMBER(st [, fmt])。函数TO_DATE(st [, fmt]) 与它们类似，将在第6章讲解。它们的任务基本上一目了然。TO_CHAR() 将一个数值变为指定格式的一个字符串值，然后可用任意字符函数操纵这个值。TO_NUMBER() 将一个字符串值变为指定格式的一个数值，然后可用数学函数操纵这个数值或将其用于数学运算。格式通常是缺省的，这样可使产生的数值或字符串与输入值具有正确的长度。

例4.34

写出一个SQL查询，以对以字符串方式存储的数进行计算。为了进行数学计算，你需要将它们转换为数值。本例将使用内部DUAL表进行演示

```
SELECT TO_NUMBER('123.45') + TO_NUMBER('234.56')
FROM DUAL;
TO_NUMBER('123.45')+TO_NUMBER('234.56')
---------------------------------------
                                 358.01
```

例4.35

写出一个SQL查询，显示社会保险号。在某个特定的表中，社会保险号为9位数(987654321)。但在此题中，要求将它们用客户的3-2-4数字格式显示（987－65－4321）。为了做到这一点，需要按以下三步操作：

a. 将数值转换为一个字符串。

b. 将字符串分隔为长度正确的三部分。

c. 将这三部分中间用连字符并置

本例将用到内部表 DUAL。对有数值的任一列名，可用（987654321）替代。

a. 用 TO_CHAR() 函数将值转换为一个字符串

```
SELECT TO_CHAR(987654321)
FROM DUAL;
TO_CHAR(9
---------
987654321
```

结果值虽然看起来无任何差异，但现在可使用字符串函数对它进行操作了。

b. 将字符串分隔为长度正确的三部分。用该结果作 SUBSTR（）函数的输入：

```
SELECT SUBSTR(TO_CHAR(987654321), 1, 3) "3",
       SUBSTR(TO_CHAR(987654321), 4, 2) "2",
       SUBSTR(TO_CHAR(987654321), 6) "4"
FROM DUAL;
3   2  4
--- -- ----
987 65 4321
```

c. 最后，将这三部分用连字符并置。

```
SELECT SUBSTR(TO_CHAR(987654321), 1, 3) || '-' ||
       SUBSTR(TO_CHAR(987654321), 4, 2) || '-' ||
       SUBSTR(TO_CHAR(987654321), 6) "SS Number"
FROM DUAL;
SS Number
-----------
987-65-4321
```

当你想控制格式时，将一个数值转换为一个字符串稍微有些复杂。表 4-8 给出了可用于 TO_CHAR() 函数的格式，能以指定方式显示数值。针对函数的格式自变量像一个模板，模板后必须跟着 9。一个 9 代表数值的每位数字。

表 4-8　TO_CHAR() 函数的格式

符号	说明
9	每个 9 代表结果中的一位数字
0	代表要显示的先导零
$	美元符号；打印在数的左边
L	任意的当地货币符号
.	打印十进制的小数点
,	打印代表千分位的逗号

例 4.36

使用内部表 DUAL，将数 123、54321 及 9874321 显示为美元值并加上逗号以增加可读性，允许值可高达百万

```
SELECT TO_CHAR(123,'$9,999,999'),
    TO_CHAR(54321,'$9,999,999'),
    TO_CHAR(9874321, '$9,999,999')
FROM DUAL;
TO_CHAR(123 TO_CHAR(543 TO_CHAR(987
----------- ------ ---- ----------
       $123     $54,321  $9,874,321
```

从结果表中可以看出，它遵循了格式模板，美元符号直接置于数的左边，逗号也放在了适当的地方。

例 4.37

以货币形式显示上述数值，并指定要显示小数位。

```
SELECT TO_CHAR(123,'$9,999,999.99'),
     TO_CHAR(54321,'$9,999,999.99'),
     TO_CHAR(9874321, '$9,999,999.99')
FROM DUAL;
TO_CHAR(123, $ TO_CHAR(54321,    TO_CHAR(987432
--- ---------- ---------------- -- ----------
       $123.00       $54,321.00  $9,874,321.00
```

例 4.38

将 1234.12345、0.4567、1.1 显示为小数点后有三位数的形式，并加上逗号增加可读性。数要小于一百万。

```
SELECT TO_CHAR(1234.12345, '999,999.999'),
     TO_CHAR(0.4567,'999,999.999'),
     TO_CHAR(1.1, '999,999.999')
FROM DUAL;
TO_CHAR(1234 TO_CHAR(0.45 TO_CHAR(1.1,
------------ ------------ ------------
   1,234.123         .457        1.100
```

注意，在适当的位置插入逗号。如果在数中的小数点后没有足够的数字，可加上 0 使小数点后具有指定的位数。如果小数点右边有过多的数字，则四舍五入，以在小数点右边显示指定的位数。

问题与答案

注意：DUAL表是一个内部表，可用于显示常数的计算结果。将在下面的一些问题中使用该表。在回答问题前，重新运行SC脚本，刷新该数据库。使用雇员表S_EMP回答下面的问题。

ID	LAST_NAME	FIRST_NAME	MAN	TITLE	DEP	SALARY
1	Martin	Carmen		President	50	4500
2	Smith	Doris	1	VP, Operations	41	2450
3	Norton	Michael	1	VP, Sales	31	2400
4	Quentin	Mark	1	VP, Finance	10	2450
5	Roper	Joseph	1	VP, Administration	50	
6	Brown	Molly	2	Warehouse Manager	41	
7	Hawkins	Roberta	2	Warehouse Manager	42	
8	Burns	Ben	2	Warehouse Manager	43	
9	Catskill	Antoinette	2	Warehouse Manager	44	
10	Jackson	Marta	2	Warehouse Manager	45	
11	Henderson	Colin	3	Sales Representative	31	
12	Gilson	Sam	3	Sales Representative	32	
13	Sanders	Jason	3	Sales Representative	33	
14	Dameron	Andre	3	Sales Representative	35	
15	Hardwick	Elaine	6	Stock Clerk	41	
16	Brown	George	6	Stock Clerk	41	
17	Washington	Thomas	7	Stock Clerk	42	
18	Patterson	Donald	7	Stock Clerk	42	
19	Bell	Alexander	8	Stock Clerk	43	
20	Gantos	Eddie	9	Stock Clerk	44	
21	Stephenson	Blaine	10	Stock Clerk	45	
22	Chester	Eddie	9	Stock Clerk	44	
23	Pearl	Roger	9	Stock Clerk	34	
24	Dancer	Bonnie	7	Stock Clerk	45	
25	Schmitt	Sandra	8	Stock Clerk	45	

4.1 表中列出的是每月工资。写出一个SQL查询，使结果表可显示所有仓库经理的工资每月增加250美元后的值：

```
SELECT last_name, first_name, salary, salary + 250
FROM s_emp
WHERE title = 'Warehouse Manager';
```

LAST_NAME	FIRST_NAME	SALARY	SALARY+250
Brown	Molly	1600	1850
Hawkins	Roberta	1650	1900
Burns	Ben	1500	1750
Catskill	Antoinette	1700	1950
Jackson	Marta	1507	1757

4.2 写出一个SQL查询，在结果表中可显示每个仓库经理目前的年工资以及每月增长250美元后的年工资。使用恰当的名称命名新列。

```
SELECT last_name, first_name, salary * 12 "CURRENT YEARLY",
(salary + 250) * 12 "NEW YEARLY"
FROM s_emp
WHERE title = 'Warehouse Manager';
```

```
LAST_NAME       FIRST_NAME          CURRENT YEARLY  NEW YEARLY
--------------- ------------------- -------------- ----------
Brown           Molly                        19200       22200
Hawkins         Roberta                      19800       22800
Burns           Ben                          18000
Catskill        Antoinette                   20400
Jackson         Marta                        18084
```

4.3 下面的SQL查询会得到什么结果表？对结果进行解释。

```
SELECT last_name,first_name,
((((salary + 250) * 12)-salary * 12)/(salary * 12)* 100)
FROM s_emp
WHERE title = 'Warehouse Manager';
```

结果表显示了每个仓库经理年工资的增长百分率。Ben Burns具有最大的增长百分率，为16.6%。Antoinette Catskill具有最小的增长百分率，为14.7%。

```
LAST_NAME   FIRST_NAME   (((SALARY+250)*12)-SALARY*12)/(SALARY*12)*100)
----------- ------------ ----------------------------------------------
Brown       Molly                                              15.625
Hawkins     Roberta                                         15.151515
Burns       Ben                                             16.666667
Catskill    Antoinette                                      14.705882
Jackson     Marta                                            16.58925
```

4.4 下面的SQL查询会得到什么？对结果进行解释。

```
SELECT last_name,first_name, salary * 12 ,
(salary * 12) + (salary * 12 * .2)
FROM s_emp
WHERE title like 'VP%';
```

结果表显示了目前副总裁的年工资，还有增加20%后的年工资。

```
LAST_NAME   FIRST_NAME      SALARY*12 (SALARY*12)+(SALARY*12*.2)
----------- --------------- --------- --------------------------
Smith       Doris               29400                      35280
Norton      Michael             28800                      34560
Quentin     Mark                29400                      35280
Roper       Joseph              30600                      36720
```

4.5　写出一个 SQL 查询，显示以下数值的绝对值：

345.2，-222，0，-1.

```
SELECT ABS(345.2), ABS(-222), ABS(0), ABS(-1)
FROM DUAL;
ABS(345.2) ABS(-222)    ABS(0)   ABS(-1)
---------- --------- --------- ---------
     345.2       222         0         1
```

4.6　下面的 SQL 查询会得到什么？对结果进行解释。

```
SELECT last_name, first_name, ROUND(salary, -2)
FROM s_emp;
```

LAST_NAME	FIRST_NAME	ROUND(SALARY,-2)
Martin	Carmen	4500
Smith	Doris	2500
Norton	Michael	2400
Quentin	Mark	2500
Roper	Joseph	2600
Brown	Molly	1600
Hawkins	Roberta	1700
Burns	Ben	1500
Catskill	Antoinette	1700
Jackson	Marta	1500
Henderson	Colin	1400
Gilson	Sam	1500
Sanders	Jason	1500
Dameron	Andre	1500
Hardwick	Elaine	1400
Brown	George	900
Washington	Thomas	1200
Patterson	Donald	800
Bell	Alexander	900
Gantos	Eddie	800
Stephenson	Blaine	900
Chester	Eddie	800
Pearl	Roger	800
Dancer	Bonnie	900
Schmitt	Sandra	1100

结果表显示了所有雇员工资和它四舍五入到最邻近的百位数的结果。

4.7　使用下面的 TRYNUM 表，回答这个问题。

FIRST	SECOND	THIRD
987	987	987
987.222	987	987.23
98.5	99	98.5

```
   98765     98765     98765
  23.987        24     23.99
  .00003         0         0
   100.9       101     100.9
  .00005         0         0
     1.9         2       1.9
    10.1        10        10
```

写出一个SQL查询，在TRYNUM表中，将FIRST列四舍五入到最邻近的十分位、百分位和千分位。

```
SELECT first, ROUND(first, 1), ROUND(first,2), ROUND(first,3)
FROM trynum;
    FIRST ROUND(FIRST,1) ROUND(FIRST,2) ROUND(FIRST,3)
--------- ------------- ------------- -------------
      987           987           987           987
  987.222         987.2        987.22       987.222
     98.5          98.5          98.5          98.5
    98765         98765         98765         98765
   23.987            24         23.99        23.987
   .00003             0             0             0
    100.9         100.9         100.9         100.9
   .00005             0             0             0
      1.9           1.9           1.9           1.9
     10.1          10.1          10.1          10.1
```

4.8 使用TRYNUM表，回答下面问题。写出一个SQL查询，在TRYNUM表中，将FIRST列四舍五入到最邻近的整数、十位数和百位数。

```
SELECT first, ROUND(first), ROUND(first, -1), ROUND(first, -2)
FROM trynum;
FIRST ROUND(FIRST) ROUND(FIRST,-1) ROUND(FIRST,-2)
--------- ------------ ------------- -------------
      987          987           990          1000
  987.222          987           990          1000
     98.5           99           100           100
    98765        98765         98770         98800
   23.987           24            20             0
   .00003            0             0             0
    100.9          101           100           100
   .00005            0             0             0
      1.9            2             0             0
     10.1           10            10             0
```

4.9 使用TRYNUM表回答问题。下面的SQL查询会得到什么？对结果进行解释。

```
SELECT second, TRUNC(second/ 25), MOD(second,25)
FROM trynum;
   SECOND TRUNC(SECOND/25) MOD(SECOND,25)
--------- ---------------- --------------
      987               39             12
      987               39             12
       99                3             24
    98765             3950             15
       24                0             24
        0                0              0
      101                4              1
        0                0              0
        2                0              2
       10                0             10
```

这个查询结果显示了在 TRYNUM 表中，每一行的 SECOND 列和它被 25 除后的结果，结果表的第 2 列显示了整数商，第 3 列显示了余数。

4.10　写出一个 SQL 查询，显示 5 及其 2 次、3 次和 4 次方的值。

```
SELECT POWER(5,1), POWER(5,2), POWER(5,3), POWER(5,4)
FROM DUAL;
POWER(5,1) POWER(5,2) POWER(5,3) POWER(5,4)
---------- ---------- ---------- ----------
         5         25        125        625
```

4.11　下面的 SQL 查询会得到什么？对结果进行解释。

```
SELECT id, salary, ROUND(salary + salary*.15, -1)
FROM s_emp
WHERE title LIKE 'VP%';
 ID     SALARY ROUND(SALARY+SALARY*.15,-1)
--- --------- ---------------------------
  2      2450                        2820
  3      2400                        2760
  4      2450                        2820
  5      2550                        2930
```

结果表显示了副总裁的月工资增加 15% 后的值，新的月工资四舍五入至最邻近的十位数。

4.12　使用 TRYNUM 表回答问题。写出一个 SQL 查询，显示 TRYNUM 表的 THIRD 列值和其整数部分被 32 除所得的余数。

```
SELECT third, MOD(TRUNC(third), 32)
FROM trynum;
    THIRD MOD(TRUNC(THIRD),32)
--------- --------------------
      987                   27
   987.23                   27
```

```
      98.5                  2
     98765                 13
     23.99                 23
         0                  0
     100.9                  4
         0                  0
       1.9                  1
        10                 10
```

4.13 写出一个SQL查询，显示S_EMP表中库存管理员的日工资。假定一个月有30天。将日工资四舍五入到最邻近的百分位，命名为“DAILY PAY”。

```
SELECT last_name, first_name, ROUND(salary/30,2) "DAILY PAY"
FROM s_emp
WHERE title = 'Stock Clerk';
LAST_NAME          FIRST_NAME           DAILY PAY
---------------- -------------------- ---------
Hardwick           Elaine                   46.67
Brown              George                   31.33
Washington         Thomas                      40
Patterson          Donald                    26.5
Bell               Alexander                28.33
Gantos             Eddie                    26.67
Stephenson         Blaine                   28.67
Chester            Eddie                    26.67
Pearl              Roger                     26.5
Dancer             Bonnie                   28.67
Schmitt            Sandra                   36.67
```

4.14 写出一个SQL查询，将所有副总裁的名字和姓氏以大写字母显示，职务以小写字母显示。

```
SELECT UPPER(first_name), UPPER(last_name), LOWER(title)
FROM s_emp
WHERE title LIKE 'VP%';
UPPER(FIRST_NAME)    UPPER(LAST_NAME)      LOWER(TITLE)
-------------------- ---- -------------- --------------------
DORIS                SMITH                 vp, operations
MICHAEL              NORTON                vp, sales
MARK                 QUENTIN               vp, finance
JOSEPH               ROPER                 vp, administration
```

4.15 写出一个SQL查询，将‘happy birthday to you !’中所有单词的第一个字符大写。使用内部表DUAL。

```
SELECT INITCAP('happy birthday to you!')
FROM DUAL;
INITCAP('HAPPYBIRTHDAY
----------------------
Happy Birthday To You!
```

记住，INITCAP（）函数对目标字符串中每一个单独的单词进行运算。

4.16　写出一个 SQL 查询，用下面的语句显示新建立的‘EMPLOYEES’列：

“The employee <id> firstinitial <last_name> is the <title>.”

```
SELECT 'The employee ' || id || ' ' || SUBSTR(first_name,1,1)
       || ' ' || last_name ||
       ' is the ' || title 'EMPLOYEES"
FROM s_emp;
EMPLOYEES
-------------------------------- ----------------------------
The employee 1 C Martin is the President
The employee 2 D Smith is the VP, Operations
The employee 3 M Norton is the VP, Sales
The employee 4 M Quentin is the VP, Finance
The employee 5 J Roper is the VP, Administration
The employee 6 M Brown is the Warehouse Manager
The employee 7 R Hawkins is the Warehouse Manager
The employee 8 B Burns is the Warehouse Manager
The employee 9 A Catskill is the Warehouse Manager
The employee 10 M Jackson is the Warehouse Manager
The employee 11 C Henderson is the Sales Representative
The employee 12 S Gilson is the Sales Representative
The employee 13 J Sanders is the Sales Representative
The employee 14 A Dameron is the Sales Representative
The employee 15 E Hardwick is the Stock Clerk
The employee 16 G Brown is the Stock Clerk
The employee 17 T Washington is the Stock Clerk
The employee 18 D Patterson is the Stock Clerk
The employee 19 A Bell is the Stock Clerk
The employee 20 E Gantos is the Stock Clerk
The employee 21 B Stephenson is the Stock Clerk
The employee 22 E Chester is the Stock Clerk
The employee 23 R Pearl is the Stock Clerk
The employee 24 B Dancer is the Stock Clerk
The employee 25 S Schmitt is the Stock Clerk
```

4.17　下面的 SQL 查询会得到什么？对结果进行解释。

```
SELECT LPAD(last_name,20),SUBSTR(first_name,1,1),
     TO_CHAR(salary,'$9,999.99')
FROM s_emp
WHERE title = 'Stock Clerk';
```

这个查询将显示所有库存管理员的姓氏（右对齐以空格填充到 20 个字符），名字的第一个字母和按指定格式显示（包括美元符号和 2 位小数）的工资。结果表如下所示：

```
LPAD(LAST_NAME,20)    S TO_CHAR(SA
--------------------- - ----------
            Hardwick E  $1,400.00
               Brown G    $940.00
          Washington T  $1,200.00
           Patterson D    $795.00
                Bell A    $850.00
              Gantos E    $800.00
          Stephenson B    $860.00
             Chester E    $800.00
               Pearl R    $795.00
              Dancer B    $860.00
              Schmitt S  $1,100.00
```

4.18 写出一个SQL查询，可打印出与问题4.17相同的图表，但要求在工资列的左边，以星号填充到15个字符。

```
SELECT LPAD(last_name,20),SUBSTR(first_name, ,1),
     LPAD(TO_CHAR(salary,'$9,999.99'),15,'*')
FROM s_emp
WHERE title = 'Stock Clerk';
LPAD(LAST_NAME,20)    S LPAD(TO_CHAR(SA
  --  ---  --- ----  - ------------ --
            Hardwick E ***** $1,400.00
               Brown G *****   $940.00
          Washington T ***** $1,200.00
           Patterson D *****   $795.00
                Bell A *****   $850.00
              Gantos E *****   $800.00
          Stephenson B *****   $860.00
             Chester E *****   $800.00
               Pearl R *****   $795.00
              Dancer B *****   $860.00
              Schmitt S ***** $1,100.00
```

注意，在每一行中只有5个星号。当你填充的列是从数值转换为字符串时，填充只延伸到格式规定的一个字符前。在此例中，格式$9,999.99具有9个字符，在左边允许有一个空格，其余5个则为星号

4.19 写出一个SQL查询，显示副总裁的职务。将每个实例中的VP用职务‘Vice President’替换，将列命名为‘VICE PRESIDENT TITLES’。

```
SELECT REPLACE(title, 'VP', 'Vice President') "VICE PRESIDENT TITLES"
FROM s_emp
WHERE title LIKE 'VP%';
VICE PRESIDENT TITLES
-  --  --  --  --- ------------------------------
Vice President, Operations
Vice President, Sales
Vice President, Finance
Vice President, Administration
```

4.20 下面的SQL查询会得到什么样的结果表？解释此结果

```
SELECT last_name, first_name
FROM s_emp
WHERE LENGTH(last_name)   5 AND LENGTH(first_name) = 5;
```

结果表显示，这个查询将打印出姓氏和名字均恰好为5个字符的人名：

```
LAST_NAME                 FIRST_NAME
------------------------- -------------------------
Smith                     Doris
Brown                     Molly
Pearl                     Roger
```

4.21 有人建立了如下名为PEOPLE的表，该表有一列包含称呼、名字和姓氏：

```
P_NAME
--------------------------
Mr. John Doe
Mrs. Susan Blake
```

写出一个SQL查询，用此列产生3个分开的列，列名分别为"TITLE"、"FIRST"和"LAST"。

- 欲解决这个问题，你需要找到位于职务和名字起始之间的第一个空格：INSTR(p_name, ' ')。
- 然后你需要找到位于名字之后和姓氏起始之间的第二个空格：INSTR(p_name,'',1,2)。
- 最后，用SUBSTR()函数打印每一列。

```
SELECT SUBSTR(p_name, 1, INSTR(p_name, ' ')) "TITLE",
SUBSTR(p_name, INSTR(p_name, ' ')+1, INSTR(p_name, ' ',2))"FIRST",
SUBSTR(p_name, INSTR(p_name, ' ',1,2)+1)"SECOND"
FROM people;
```

结果表显示该查询达到了预期目的：

```
TITLE                FIRST                SECOND
-------------------- -------------------- --------------------
Mr.                  John                 Doe
Mrs.                 Susan                Blake
```

4.22 写出一个SQL查询，打印ID是奇数的雇员的ID、名字和姓氏。记住，ID是作为字符数据存储的。先将它们转换为数值数据，然后用MOD函数找到奇数（如果一个数的MOD 2正好为1，该数为奇数）。这些函数将出现在语句的WHERE子句中。

```
SELECT id, first_name, last_name
FROM s_emp
WHERE MOD(TO_NUMBER(id),2) = 1;
ID  FIRST_NAME           LAST_NAME
--- -------------------- --------------------
1   Carmen               Martin
3   Michael              Norton
5   Joseph               Roper
7   Roberta              Hawkins
9   Antoinette           Catskill
```

```
11  Colin                Henderson
13  Jason                Sanders
15  Elaine               Hardwick
17  Thomas               Washington
19  Alexander            Bell
21  Blaine               Stephenson
23  Roger                Pearl
25  Sandra               Schmitt
```

4.23 有人为药店建立了一个小表，对不同药的不同税率进行了估计。但是，对于只有2.5%税的药品，表中未列出其税率。表如下所示：

```
ITEM                      COST       TAX
------------------------- ---------- ----------
bandages                  1.52       .05
aspirin                   2.55
candy                      .79       .06
antacid                   2.34
```

写出一个 SQL 查询，可打印这些药品包括税在内的总成本。如果未给出税率，就定为 2.5%。将所得值四舍五入到最邻近的百分位。

```
SELECT item, cost, NVL(tax,0.025),
     ROUND(cost + cost * NVL(tax,0.025),2) "TOTAL"
FROM store;
ITEM                      COST NVL(TAX,0.025)      TOTAL
------------------------- ---------- ------------- ---------
bandages                  1.52              .05       1.6
aspirin                   2.55             .025      2.61
candy                      .79              .06       .84
antacid                   2.34             .025       2.4
```

4.24 重复上述问题。此次要求用美元符号显示金额字段。如果不到 1 美元时，前面加上 0。

```
SELECT item, TO_CHAR(cost,'$0.99'), TO_CHAR(NVL(tax,0.025),'$0.99'),
    TO_CHAR(ROUND(cost + cost * NVL(tax,0.025),2),'$0.99') "TOTAL"
FROM store;
ITEM                 TO_CHA TO_CHA TOTAL
-------------------- ------ -- --- ------
bandages              $1.52  $0.05  $1.60
aspirin               $2.55  $0.03  $2.61
candy                 $0.79  $0.06  $0.84
antacid               $2.34  $0.03  $2.40
```

补 充 题

注意：重新运行 SG 脚本以刷新数据库，使用 S_ORD 表回答下面的问题。

```
CUS DATE_ORDE DATE_SHIP SAL     TOTAL PAYMEN
--- --------- --------- --- --------- ------
100 31-AUG-92 10-SEP-92 11    601100 CREDIT
101 31-AUG-92 15-SEP-92 14    8056.6 CREDIT
102 01-SEP-92 08-SEP-92 15      8335 CREDIT
103 02-SEP-92 22-SEP-92 15       377 CASH
104 03-SEP-92 23-SEP-92 15     32430 CREDIT
105 04-SEP-92 18-SEP-92 11   2722.24 CREDIT
106 07-SEP-92 15-SEP-92 12     15634 CREDIT
107 07-SEP-92 21-SEP-92 15    142171 CREDIT
108 07-SEP-92 10-SEP-92 13    149570 CREDIT
109 08-SEP-92 28-SEP-92 11   1020935 CREDIT
110 09-SEP-92 21-SEP-92 11   1539.13 CASH
111 09-SEP-92 21-SEP-92 11      2770 CASH
97  28-AUG-92 17-SEP-92 12     84000 CREDIT
98  31-AUG-92 10-SEP-92 14       595 CASH
99  31-AUG-92 18-SEP-92 14      7707 CREDIT
112 31-AUG-92 10-SEP-92 12       550 CREDIT
```

4.25　写出一个 SQL 查询，显示每一个定单从订货到运货之间的天数。

4.26　每一个定单的总额不包括税。写出一个 SQL 查询，显示每一个定单的总额、5% 的销售税和包括税在内的付款总额。恰当命名新列。

4.27　公司有一条规定：在八月份，任何总额超过 7000 美元的定单，均可获得 500 美元的打折。写出一个 SQL 查询，显示这些定单在包括 5% 的销售税后的总额。恰当命名新列。

4.28　重新写出上一题的 SQL 查询，将总额显示为四舍五入到最邻近的美元数。

4.29　写出一个 SQL 查询，可打印总额、将总额截断到最邻近的 100 美元的值、每个定单截断后的总额与 6000 美元之差的绝对值。恰当命名新列。

4.30　写出一个 SQL 查询，打印出 10 的前 5 次方。（提示：使用 DUAL 表）

4.31　假设总帐单以现金方式支付，而现金都是面值为 100 美元的钞票。写出一个 SQL 查询，显示每个定单将收到多少张百元钞票以及在付此百元钞票的基础上，还应付多少现金？恰当命名新列。

使用如下 WORLD_CITIES 表的子集，回答问题 4.32 至 4.37。

```
CITY                 COUNTRY              CONTINENT
-------------------- -------------------- -------------
ATHENS               GREECE               EUROPE
ATLANTA              UNITED STATES        NORTH AMERICA
DALLAS               UNITED STATES        NORTH AMERICA
NASHVILLE            UNITED STATES        NORTH AMERICA
VICTORIA             CANADA               NORTH AMERICA
PETERBOROUGH         CANADA               NORTH AMERICA
VANCOUVER            CANADA               NORTH AMERICA
TOLEDO               UNITED STATES        NORTH AMERICA
```

WARSAW	POLAND	EUROPE
LIMA	PERU	SOUTH AMERICA
RIO DE JANEIRO	BRAZIL	SOUTH AMERICA
SANTIAGO	CHILE	SOUTH AMERICA
BOGOTA	COLOMBIA	SOUTH AMERICA
BUENOS AIRES	ARGENTINA	SOUTH AMERICA
QUITO	ECUADOR	SOUTH AMERICA
CARACAS	VENEZUELA	SOUTH AMERICA
MADRAS	INDIA	ASIA
NEW DELHI	INDIA	ASIA
BOMBAY	INDIA	ASIA
MANCHESTER	ENGLAND	EUROPE
LONDON	ENGLAND	EUROPE
MOSCOW	RUSSIA	EUROPE
PARIS	FRANCE	EUROPE
SHENYANG	CHINA	ASIA
CAIRO	EGYPT	AFRICA
TRIPOLI	LIBYA	AFRICA
BEIJING	CHINA	ASIA
ROME	ITALY	EUROPE
TOKYO	JAPAN	ASIA
SYDNEY	AUSTRALIA	AUSTRALIA
SPARTA	GREECE	EUROPE
MADRID	SPAIN	EUROPE

4.32 WORLD_CITIES表都以大写字母存储。写出一个SQL查询，以首字母大写、其余字母小写的方式显示在北美及南美的城市。

4.33 写出一个SQL查询，显示除了北美和南美外，位于其他洲的所有城市。要求显示以下两列：第一列列名为“PLACE”，包括城市和国家，要求首字母大写，中间以逗号隔开。第二列是洲，全以小写方式显示。

4.34 写出一个SQL查询，显示位于欧洲的城市和国家，将欧洲的英文字母的后三个字母截去，显示为EUR。

4.35 写出一个SQL查询，显示位于欧洲的城市。将城市名中的全部S用美元符号代替。

4.36 写出一个SQL查询，打印出在北美和南美的城市和国家。将洲显示为“N America”或“S America”。将所有列以首字符大写的方式给出。

4.37 写出一个SQL查询，显示城市名和国家名具有同样多字母的城市和国家。

4.38 写出一个SQL查询，显示S_ORD表中客户ID为偶数的客户ID和付款总额。按客户ID的顺序列出。

4.39 重复上一个问题，此次，将付款总额四舍五入到邻近的十位数，并且在左边用星号填充至15个字符宽。

4.40 有一个名为ADDIT（如下所示）、具有两列数的表，其中有些值为NULL。写出一个SQL查询，显示这两列数及它们的总和。将NULL值用0取代。

```
     FIRST     SECOND
---------- ----------
   4567.88      23.33
    345.34
              2554.23
   2.99876       4.34
```

补充题答案

4.25

```
SELECT id, date_ordered, date_shipped,
date_shipped-date_ordered
FROM s_ord;
ID  DATE_ORDE DATE_SHIP DATE_SHIPPED-DATE_ORDERED
--- --------- --------- ------------------------
100 31-AUG-92 10-SEP-92                       10
101 31-AUG-92 15-SEP-92                       15
102 01-SEP-92 08-SEP-92                        7
103 02-SEP-92 22-SEP-92                       20
104 03-SEP-92 23-SEP-92                       20
105 04-SEP-92 18-SEP-92                       14
106 07-SEP-92 15-SEP-92                        8
107 07-SEP-92 21-SEP-92                       14
108 07-SEP-92 10-SEP-92                        3
109 08-SEP-92 28-SEP-92                       20
110 09-SEP-92 21-SEP-92                       12
111 09-SEP-92 21-SEP-92                       12
97  28-AUG-92 17-SEP-92                       20
98  31-AUG-92 10-SEP-92                       10
99  31-AUG-92 18-SEP-92                       18
112 31-AUG-92 10-SEP-92                       10
```

4.26

```
SELECT id, total, total * .05 "TAX",
total + (total * .05) "TOTAL BILL"
FROM s_ord;
ID  TOTAL          TAX TOTAL BILL
--- ----- ---------- -----------
100 601100      30055      631155
101  8056.6     402.83     8459.43
102    8335     416.75     8751.75
103     377      18.85      395.85
104   32430     1621.5     34051.5
105 2722.24     136.112    2858.352
106   15634      781.7     16415.7
107  142171    7108.55    149279.55
108  149570     7478.5     157048.5
109 1020935   51046.75   1071981.8
110 1539.13    76.9565    1616.0865
111    2770      138.5      2908.5
97    84000       4200       88200
98      595      29.75      624.75
99     7707     385.35     8092.35
112     550       27.5       577.5
```

4.27

```
SELECT id,date_ordered, total- 500, (total - 500)* .05 "TAX",
(total - 500) + ((total - 500) * .05) "TOTAL BILL"
FROM s_ord
WHERE date_ordered < TO_DATE('01-SEP-1992','DD-MON-YYYY') AND
      date_ordered >= TO_DATE('01-AUG-1992','DD-MON-YYYY')
      AND total >= 7000;
```

```
ID  DATE_ORDE TOTAL-500       TAX TOTAL BILL
--- --------- --------- --------- ----------
100 31-AUG-92    600600     30030     630630
101 31-AUG-92    7556.6    377.83    7934.43
97  28-AUG-92     83500      4175      87675
99  31-AUG-92      7207    360.35    7567.35
```

与比较日期有关的更多帮助，参见第 6 章。

4.28

```
SELECT id,date_ordered, total- 500, (total - 500)* .05 "TAX",
ROUND(((total - 500) + (total - 500) * .05)) "TOTAL BILL"
FROM s_ord
WHERE date_ordered < TO_DATE('01-SEP-1992','DD-MON-YYYY')
      AND date_ordered >= TO_DATE('01-AUG-1992','DD-MON-YYYY')
      AND total >= 7000;
```

```
ID  DATE_ORDE TOTAL-500       TAX TOTAL BILL
--- --------- --------- --------- ----------
100 31-AUG-92    600600     30030     630630
101 31-AUG-92    7556.6    377.83      7934
97  28-AUG-92     83500      4175      87675
99  31-AUG-92      7207    360.35      7567
```

4.29

```
SELECT total, TRUNC(total, -2), ABS(TRUNC(total, -2)- 6000)
FROM s_ord;
```

```
     TOTAL TRUNC(TOTAL,-2) ABS(TRUNC(TOTAL,-2)-6000)
---------- --------------- -----------------------
    601100          601100                  595100
    8056.6            8000                    2000
      8335            8300                    2300
       377             300                    5700
     32430           32400                   26400
   2722.24            2700                    3300
     15634           15600                    9600
    142171          142100                  136100
    149570          149500                  143500
   1020935         1020900                 1014900
   1539.13            1500                    4500
      2770            2700                    3300
     84000           84000                   78000
       595             500                    5500
      7707            7700                    1700
       550             500                    5500
```

4.30

```
SELECT POWER(10,1), POWER(10,2), POWER(10,3), POWER(10,4),
POWER(10,5)
FROM DUAL;
POWER(10,1) POWER(10,2) POWER(10,3) POWER(10,4) POWER(10,5)
----------- ----------- ----------- ----------- -----------
         10         100        1000       10000      100000
```

4.31

```
SELECT total, TRUNC(total/100) "HUNDREDS",
     MOD(total, 100) "OTHER CASH"
FROM s_crd;
    TOTAL  HUNDREDS  OTHER CASH
--------- --------- ----------
   601100      6011           0
   8056.6        80        56.6
     8335        83          35
      377         3          77
    32430       324          30
  2722.24        27       22.24
    15634       156          34
   142171      1421          71
   149570      1495          70
  1020935     10209          35
  1539.13        15       39.13
     2770        27          70
    84000       840           0
      595         5          95
     7707        77           7
      550         5          50
```

4.32

```
SELECT INITCAP(city),INITCAP(country),INITCAP(continent)
FROM world_cities
WHERE continent LIKE '%AMERICA';
INITCAP(CITY)        INITCAP(COUNTRY)     INITCAP(CONTINENT)
-------------------- -------------------- ------------------
Atlanta              United States        North America
Dallas               United States        North America
Nashville            United States        North America
Victoria             Canada               North America
Peterborough         Canada               North America
Vancouver            Canada               North America
Toledo               United States        North America
Lima                 Peru                 South America
Rio De Janeiro       Brazil               South America
Santiago             Chile                South America
Bogota               Colombia             South America
Buenos Aires         Argentina            South America
Quito                Ecuador              South America
Caracas              Venezuela            South America
```

4.33

```
SELECT INITCAP(city) || ', ' || INITCAP(country) "PLACE",
     LOWER(continent)
FROM world_cities
WHERE NOT continent LIKE '%AMERICA';
```

```
PLACE                                                    LOWER(CONTINENT)
-------- ------------------------------------------------ ----------
Athens, Greece                                           europe
Warsaw, Poland                                           europe
Madras, India                                            asia
New Delhi, India                                         asia
Bombay, India                                            asia
Manchester, England                                      europe
London, England                                          europe
Moscow, Russia                                           europe
Paris, France                                            europe
Shenyang, China                                          asia
Cairo, Egypt                                             africa
Tripoli, Libya                                           africa
Beijing, China                                           asia
Rome, Italy                                              europe
Tokyo, Japan                                             asia
Sydney, Australia                                        australia
Sparta, Greece                                           europe
Madrid, Spain                                            europe
```

4.34

```
SELECT city, country, RTRIM(continent, 'OPE') "TRIMMED"
FROM world_cities
WHERE continent = 'EUROPE';
```

```
CITY                      COUNTRY                   TRIMMED
------------------------- ------------------------- ---------
ATHENS                    GREECE                    EUR
WARSAW                    POLAND                    EUR
MANCHESTER                ENGLAND                   EUR
LONDON                    ENGLAND                   EUR
MOSCOW                    RUSSIA                    EUR
PARIS                     FRANCE                    EUR
ROME                      ITALY                     EUR
SPARTA                    GREECE                    EUR
MADRID                    SPAIN                     EUR
```

4.35

```
SELECT REPLACE(city, 'S', '$')
FROM WORLD_CITIES
WHERE continent = 'EUROPE';
REPLACE(CITY,'S','$')
-------------------------
ATHEN$
WAR$AW
MANCHE$TER
LONDON
MO$COW
PARI$
ROME
$PARTA
MADRID
```

4.36

```
SELECT INITCAP(city), INITCAP(country) "COUNTRY",
     SUBSTR(continent, 1, 1)|| ' ' ||
     INITCAP(SUBSTR(continent, INSTR(continent, ' ')+1))
     "CONTINENT"
FROM WORLD_CITIES
WHERE continent LIKE '%AMERICA';
```

INITCAP(CITY)	COUNTRY	CONTINENT
Atlanta	United States	N America
Dallas	United States	N America
Nashville	United States	N America
Victoria	Canada	N America
Peterborough	Canada	N America
Vancouver	Canada	N America
Toledo	United States	N America
Lima	Peru	S America
Rio De Janeiro	Brazil	S America
Santiago	Chile	S America
Bogota	Colombia	S America
Buenos Aires	Argentina	S America
Quito	Ecuador	S America
Caracas	Venezuela	S America

4.37

```
SELECT city, country, LENGTH(city)
FROM world_cities
WHERE LENGTH(city) = LENGTH(country);
```

CITY	COUNTRY	LENGTH(CITY)
ATHENS	GREECE	6
WARSAW	POLAND	6
LIMA	PERU	4

```
MOSCOW                    RUSSIA                              6
CAIRO                     EGYPT                               5
TOKYO                     JAPAN                               5
SPARTA                    GREECE                              6
```

4.38

```
SELECT customer_id, total
FROM s_ord
WHERE MOD(TO_NUMBER(customer_id),2) = 0
ORDER BY customer_id;
CUS      TOTAL
--- ---------
202        595
204     601100
204       2770
206       8335
208        377
208      32430
210      15634
210        550
212     149570
214    1539.13
```

4.39

```
SELECT customer_id, LPAD(TO_CHAR(ROUND(total, 1),
'9,999'),15,'*')
FROM s_ord
WHERE MOD(TO_NUMBER(customer_id),2) = 0
ORDER BY customer_id;
CUS LPAD(TO_CHAR(RO
--- --------------- ------
202 *********   600
204 *********£££££
204 ********* 2,770
206 ********* 8,340
208 *********   380
208 *********£££££
210 *********£££££
210 *********   550
212 *********£££££
214 ********* 1,540
```

注意，如果一个数太长，它不可被截断。数字以英镑符号取代。

4.40

```
SELECT NVL(first,0), NVL(second,0), NVL(first,0)+NVL(second,0)
FROM addit;
NVL(FIRST,0) NVL(SECOND,0) NVL(FIRST,0)+NVL(SECOND,0)
------------ ------------ ----------------------------
     4567.88        23.33                     4591.21
      345.34            0                      345.34
           0      2554.23                     2554.23
     2.99876         4.34                     7.33876
```

第5章 分组函数

5.1 分组函数概述

前一章说明了算术运算符的使用以及用数据库中单个值做参数的内部函数。在数据库中，同时检查或者操纵多行数据常常是非常重要的。例如，在雇员表中，可能要查雇员的最高工资、平均工资或者销售代表人数等。本章重点讲分组函数。分组函数有时也称为聚集函数。这些函数的操作对象是数据库表中行的集合，产生的结果是一个值。分组函数列在表5-1中。

表5-1 分组函数

函数名	输入参数	返回值
AVG([DISTINCT | ALL] n)	n=列名	列n的平均值
COUNT([ALL] *)	无	返回查询范围内的行数包括重复值和空值
COUNT([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式为非空值的行数
MAX([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式的最大值
MIN([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式的最小值
STDEV([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式的标准偏差，忽略空值
SUM([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式值的总和
VARIANCE([DISTINCT | ALL] n)	n=列名或表达式	该列或表达式的方差，忽略空值

注意函数参数表中的关键字DISTINCT和ALL。单词ALL的含义是检验表中的所有行。在SQL查询中，因为ALL是缺省设置，所以该单词通常并不包含在SQL查询语句中。单词DISTINCT的含义是在计算前将重复的值从列中去掉。如果需要这样做的话，单词DISTINCT必须包含在语句中。在后面的COUNT()函数一节中，将会用实例来说明DISTINCT的用法。一般情况下，在大多数商业应用中并不需要函数STDEV()和VARIANCE()。本书不讨论这两个函数，但是假如需要的话，还是可采用的。

如果需要恢复数据库，再运行SC脚本即可。表5-2中的数据是订货清单信息表S_ITEM的子集，在随后的各节中将使用它。

表 5-2 ITEM 表

ORD	ITEM_ID	PRODUCT	PRICE	QUANTITY	QUANTITY_SHIPPED
100	1	10011	135	500	500
100	2	10013	380	400	400
100	3	10021	14	500	500
100	5	30326	582	600	600
100	7	41010	8	250	250
100	6	30433	20	450	450
100	4	10023	36	400	400
101	1	30421	16	15	15
101	3	41010	8	20	20
101	5	50169	4.29	40	40
101	6	50417	80	27	27
101	7	50530	45	50	50
101	4	41100	45	35	35
101	2	40422	50	30	30

5.2 SUM(n)和AVG(n)函数

SUM(n)和 AVG(n)函数分别返回指定列中所有值的和及平均值,它们都忽略空值。

例 5.1

写一个 SQL 查询，显示各项订货价格的总和及平均值。

```
SELECT SUM(price) "SUM", AVG(price) "AVERAGE"
FROM s_item;
```

在结果表中，显示了每一个函数返回的值仅仅有一个。

```
       SUM    AVERAGE
--------- ---------
  1423.29 101.66357
```

例 5.2

这些函数也可以用于表的子集。写出一个 SQL 查询，显示定单号为 100 的订购产品的价格总和以及价格的平均值。

```
SELECT SUM(price) "SUM", AVG(price) "AVERAGE"
FROM s_item
WHERE ord_id = '100';
       SUM    AVERAGE
--------- ---------
      1175 167.85714
```

在表5-3中，表示的是另一个表STATS。在下面的许多例子中，将使用该表说明这些函数如何处理有空值的列。

表5-3 STATS表

EVEN	ODD
2	1
	3
6	5
8	
10	9
	11
14	
16	15

例5.3

写一个SQL查询，显示在STATS表中偶数之和及平均值。

```
SELECT SUM(even) "SUM", AVG(even) "AVERAGE"
FROM stats;
       SUM    AVERAGE
---------- ----------
        56 9.3333333
```

这个例子说明，空值并没有假定为0。这两个函数都忽略了空值。

5.3 MAX(n)和MIN(n)函数

MAX(n)和MIN(n)函数返回指定列中的最大值和最小值。这两个函数适用于所有数据类型。

例5.4

写一个SQL查询，显示表S_ITEM中任意产品的最高价和最低价。

```
SELECT MAX(price) "MAX", MIN(price) "MIN"
FROM s_item;
       MAX        MIN
---------- ----------
       582       4.29
```

把查询得到的结果表与表5-2的原始数据进行比较，可以看到最高价是从定单号100中得到的，最低价的项目是从定单号101得到的。然而，有时需要仅从一个指定列的子集得到最大值或者最小值。下面的例子说明了如何做到这一点。

例5.5

写一个SQL查询，显示在表S_ITEM的定单号101中，任意产品的最高价和最低价。

```
SELECT MAX(price) "MAX", MIN(price) "MIN"
FROM s_item
WHERE ord_id = '101';
       MAX        MIN
--------- ---------
        80       4.29
```

例5.6

MAX(n)和MIN(n)函数也忽略了空值。写一个SQL查询，显示在表STATS中，ODD列的最大值与最小值。

```
SELECT MAX(odd) "MAX", MIN(odd) "MIN"
FROM stats;
       MAX        MIN
--------- ---------
        15          1
```

注意，这里的空值被忽略了。它们并没有把空值假定为0，否则0应作为最小值。

例5.7

写一个SQL查询，显示最大或最小的产品号。

```
SELECT MAX(product_id) "MAX", MIN(product_id) "MIN"
FROM s_item;
       MAX        MIN
--------- ---------- -
     50530      10011
```

在S_ITEM表中，product_id的数据类型是字符串。这个查询说明了MAX()和MIN()也适用于字符串。假如这些值是姓名，如“THOMAS”和“ALIEN”，MAX()将返回“THOMAS”，因为在字母表中，“THOMAS”在“ALLEN”后面。在混合数据类型情况下，用这些函数要非常小心。字符将按照它的ASCII码值计算，而且所有的大写字母比所有的小写字母都小。

5.4 COUNT()函数

这些COUNT()函数非常相似，很容易引起混淆。它们都返回一个特定的数，准确地理解每个函数的计数内容是很重要的。COUNT()函数适用于任何数据类型的列。

5.4.1 COUNT(*)函数

COUNT(*)函数是惟一一个以“*”作为参数的函数。它对整个表的所有行计数，或者对被查询指定的子集计数，它忽略任意列的空值。

例5.8

写一个SQL查询，显示在表STATS中的行数。

```
SELECT COUNT(*) "NUMBER OF ROWS"
FROM stats;
NUMBER OF ROWS
--------------
             8
```

这个例子说明，该函数忽略了任一列的空值。下一个例子说明该函数将返回被查询指定的表的子集中的行数。

例5.9

写一个SQL查询，显示在表S_ITEM中，定单号是100的行数。

```
SELECT COUNT(*) "ORDER 100"
FROM s_item
WHERE ord_id = '100';
ORDER 100
---------
        7
```

5.4.2 COUNT(ALL n)和COUNT(n)函数

COUNT(ALL n)和COUNT(n)函数通常是等效的。它们显示存在指定列n中已知值的个数。也可以读作“显示列n中已知值的个数”。关键字ALL指明在计数时查找所有的列。请看下一个例子，你会相信两个函数返回的是相同的值

例5.10

写一个SQL查询，显示在表STATS的EVEN列中值的个数

```
SELECT COUNT(ALL even) "WITH ALL", COUNT(even) "WITHOUT ALL"
FROM stats;
 WITH ALL  WITHOUT ALL
---------- -----------
         6           6
```

5.4.3 COUNT(DISTINCT n) 函数

在COUNT函数中，用关键字DISTINCT返回存在指定列n中不同值的个数。该函数适用于任意数据类型的数据。注意，关键字DISTINCT或ALL与列名之间没有逗号。

例5.11

写一个SQL查询，显示表5-2列出的S_ITEM表中数据的总行数、产品号数和定单号数。

```
SELECT COUNT(*) "TOTAL", COUNT(DISTINCT product_id) "PRODUCTS",
COUNT(DISTINCT ord_id) "ORDERS"
FROM s_item;
    TOTAL   PRODUCTS     ORDERS
--------- --------- ---------
       14        13          2
```

仔细地检查原有的表，可以发现产品号41010出现了两次。该查询的结果表明，在两个定单中，总行数是14，有13种不同的产品被订购。

5.5 单一值和分组函数的结合

在第4章中，讨论的是算术运算符和内部函数对单一值的操作，这些操作可以跟本章的分组函数相结合。例如，可能需要知道在表S_ITEM中定单号100的合计的平均值。为找到每项的合计，用数量乘单价，即（PRICE * QUANTITY)。为了查到整个定单号100的平均值，需要在同一查询中用AVG()函数。

例5.12

写一个SQL查询，显示定单号100中所有订货的合计平均值。

```
SELECT AVG(price * quantity) "AVERAGE TOTAL OF ORDER 100"
FROM s_item
WHERE ord_id = '100';
AVERAGE TOTAL OF ORDER 100
--------------------------
                1150.9429
```

SQL允许内部函数的任意组合。可以使用算术运算符对分组函数的结果进行操作。

例5.13

写一个SQL查询，显示定单号100中所有订货价格合计的范围。或者换一句话说，显

示合计的最大值与最小值之差。

```
SELECT MAX(price*quantity)-MIN(price*quantity) "COST DIFFERENCE"
FROM s_item
WHERE ord_id = '100';
COST DIFFERENCE
---------------
         347200
```

5.6 显示指定组的信息

用该查询的 WHERE 子句可以考虑选择单独的一些行，用 ORDER 子句可以使查询的结果按指定的列排序显示，两个分组运算符 GROUP BY 和 HAVING 可以实现类似的任务，但却是对分组而非单个的一些值。查询语句的语法如下：

```
SELECT column_list
FROM tablename
WHERE condition_list
GROUP BY column_list HAVING condition_list;
```

5.6.1 GROUP BY 子句

在上节中，例 5.12 和例 5.13 表明在定单号 100 中价格的平均值和各项订货价格的范围，可以使用 GROUP BY 子句查询所有定单的价格平均值和价格的范围。

例 5.14

写一个 SQL 查询，显示每个定单中所有订货价格合计的平均值和价格合计的范围。

```
SELECT ord_id, AVG(price * quantity) "AVERAGE TOTAL OF ORDER",
     MAX(price*quantity)-MIN(price*quantity) "COST DIFFERENCE"
FROM s_item
GROUP BY ord_id;
ORD AVERAGE TOTAL OF ORDER COST DIFFERENCE
--- ---------------------- ---------- --
100                85871.429          347200
101                1150.9429            2090
```

注意，在 SELECT 语句中，AVG()、MAX() 和 MIN() 分组函数像单独的列一样被列出。通常在一个 SELECT 语句中，混合单独的列和分组项是不可能的。然而，GROUP BY 指示要操作的单独的各组。为了清楚起见，将分组的定单号的值，像上面的 100 和 101 的列名 (ord_id) 包含在显示的列名中是很重要的。在查询的结果图中，清楚地列出了每张定单的合计平均值和差额。

通过在 GROUP BY 子句中包含一个以上的列名，可以显示各组及子组的概要信息。通过使用 PROGRAMMER 表的子集（数据在表 5-4 中）来说明这个问题。

表 5-4 PROGRAMMER 表

EMP	LAST_NAME	HIRE_DATE	LANGUAGE	CLEARANCE
201	Campbell	01-JAN-95	VB	Secret
390	Bell	01-MAY-93	Java	Top Secret
789	Hixon	31-AUG-98	VB	Secret
134	McGurn	15-JUL-95	C++	Secret
896	Sweet	15-JUN-97	Java	Top Secret
345	Rowlett	15-NOV-99	Java	
563	Reardon	15-AUG-94	C++	Confidential

例 5.15

假定你需要知道每类许可证有多少程序设计员，使用各种程序设计语言的程序设计员有多少。你应该先写两个分开的查询：(a) 按程序设计语言分组显示；(b) 按许可证分组显示。

(a)

```
SELECT language, COUNT(*) "IN EACH LANGUAGE"
FROM programmer
GROUP BY language;
LANGUAGE        IN EACH LANGUAGE
--------------- ----------------
C++                            2
Java                           3
VB                             2
```

(b)

```
SELECT clearance, COUNT(*) "IN EACH CLEARANCE"
FROM programmer
GROUP BY clearance;
CLEARANCE                IN EACH CLEARANCE
------------ ----------------
Confidential                             1
Secret                                   3
Top Secret                               2
                                         1
```

注意，许可证的空值在各自的范畴里都被COUNT(*) 函数考虑在内了。在这个例子中，COUNT(*) 函数返回了每个类目内的行数。

(c)

这两个查询回答了上述问题。但有什么办法将它们组合成一个表并显示在每种许可证中可采用的程序设计语言呢？是的，该查询可以写成：

```
SELECT language, clearance,
    COUNT(*) "IN EACH CLEARANCE"
FROM programmer
GROUP BY language, clearance;
```

```
LANGUAGE        CLEARANCE                    IN EACH CLEARANCE
--------------- ---------------------------- -----------------
C++             Confidential                                 1
C++             Secret                                       1
Java            Top Secret                                   2
Java                                                         1
VB              Secret                                       2
```

5.6.2 HAVING 子句

正如 WHERE 子句限制了 SELECT 语句显示的行数一样，跟在 GROUP BY 子句后的 HAVING 子句限制了显示的组数。

例 5.16

利用例 5.15 的信息写一个查询，仅显示每种许可证中有 1 个以上程序员使用的程序设计语言

```
SELECT language, clearance,
    COUNT(*) "IN EACH CLEARANCE"
FROM programmer
GROUP BY language, clearance
HAVING COUNT(*) > 1;
```

```
LANGUAGE        CLEARANCE                    IN EACH CLEARANCE
--------------- ---------------------------- -----------------
Java            Top Secret                                   2
VB              Secret                                       2
```

在 HAVING 内不仅能使用COUNT(*) 函数，任何分组函数都能使用。当要判断每个值时，用 WHERE 子句；当要判断每个分组函数的值时，用 HAVING 子句。这一点是很重要的。

例 5.17

在表 S_ITEM 中，根据在表 5-2 中的数据，显示每张定单总额（PRICE * QUANTI-

TY）大于1000美元的定单的产品数。这个查询将要使用WHERE子句，因为要对每张定单的每项订货进行检查。

```
SELECT ord_id, COUNT(*) "MORE THAN $1000"
FROM s_item
WHERE price*quantity > 1000
GROUP BY ord_id;
ORD MORE THAN $1000
--- ---------------
100                7
101                4
```

在这个例子中用了WHERE子句，因为要限制的是单个的值，或者单价乘数量的结果。当对分组函数的值进行限制时，使用HAVING子句。

例5.18

写一个SQL查询，显示定单平均价格大于2000美元的产品数。

```
SELECT ord_id, AVG(price * quantity) "AVG",
       COUNT(*) "AVG MORE THAN $2000"
FROM s_item
GROUP BY ord_id
HAVING AVG(price * quantity) > 2000;
ORD         AVG AVG MORE THAN $2000
--- --------- -------------------
100 85871.429                   7
```

在这个例子中，AVG()的函数值限制了COUNT()函数的返回值。记住，WHERE子句通过检查每个单独的值限制显示的行数。HAVING子句通过检查每个分组函数的值限制显示结果的行数。

问题与答案

注意：回答这些问题之前，重新运行SC脚本以刷新数据库。在这节的问题中，我们使用雇员表S_EMP。

```
ID  LAST_NAME       FIRST_NAME      MAN TITLE                DEP    SALARY
--- --------------- --------------- --- -------------------- --- --------
1   Martin          Carmen              President            50       4500
2   Smith           Doris           1   VP, Operations       41       2450
3   Norton          Michael         1   VP, Sales            31       2400
4   Quentin         Mark            1   VP, Finance          10       2450
5   Roper           Joseph          1   VP, Administration   50       2550
6   Brown           Molly           2   Warehouse Manager    41       1600
```

7	Hawkins	Roberta	2	Warehouse Manager	42	1650
8	Burns	Ben	2	Warehouse Manager	43	1500
9	Catskill	Antoinette	2	Warehouse Manager	44	1700
10	Jackson	Marta	2	Warehouse Manager	45	1507
11	Henderson	Colin	3	Sales Representative	31	1400
12	Gilson	Sam	3	Sales Representative	32	1490
13	Sanders	Jason	3	Sales Representative	33	1515
14	Dameron	Andre	3	Sales Representative	35	1450
15	Hardwick	Elaine	6	Stock Clerk	41	1400
16	Brown	George	6	Stock Clerk	41	940
17	Washington	Thomas	7	Stock Clerk	42	1200
18	Patterson	Donald	7	Stock Clerk	42	795
19	Bell	Alexander	8	Stock Clerk	43	850
20	Gantos	Eddie	9	Stock Clerk	44	800
21	Stephenson	Blaine	10	Stock Clerk	45	860
22	Chester	Eddie	9	Stock Clerk	44	800
23	Pearl	Roger	9	Stock Clerk	34	795
24	Dancer	Bonnie	7	Stock Clerk	45	860
25	Schmitt	Sandra	8	Stock Clerk	45	1100

5.1 写一个SQL查询，显示月工资的总和及平均值。

```
SELECT SUM(salary) "SUM", AVG(salary) "AVG"
FROM s_emp;
    SUM          AVG
--------- ---------
  38562     1542.48
```

5.2 写一个SQL查询，显示在表中描述的不同的部门数并与表中行数进行比较。

```
SELECT COUNT(DISTINCT dept_id) "DEPTS", COUNT(*) "ROWS"
FROM s_emp;
    DEPTS       ROWS
--------- ---------
       12         25
```

5.3 写一个SQL查询，显示在表中描述的不同的经理ID的个数并与表中行数进行比较。

```
SELECT COUNT(DISTINCT manager_id) "MANAGER IDS",
COUNT(*) "ROWS"
FROM s_emp;
MANAGER IDS       ROWS
----------- ---------
          8         25
```

注意，在该表中，总裁没有经理ID。这个空值并没有包含在不同的经理ID的计数中，

然而它包含在该表行数计数的总和中了。

5.4 写一个 SQL 查询，显示库存管理员的平均工资。

```
SELECT AVG(salary) "AVERAGE"
FROM s_emp
WHERE title = 'Stock Clerk';
  AVERAGE
---------
945.45455
```

5.5 写一个 SQL 查询，显示仓库经理们的最高工资和最低工资。

```
SELECT MAX(salary) "MAX", MIN(salary) "MIN"
FROM s_emp
WHERE title = 'Warehouse Manager';
       MAX        MIN
---------- ----------
      1700       1500
```

5.6 这个 SQL 查询显示的是什么？解释之。

```
SELECT MAX(last_name), MIN(last_name),
     COUNT(*), COUNT(DISTINCT last_name)
FROM s_emp;
```

这些函数能用在数据类型是字符串的列上。下面的结果表给出了按字母表最后出现的雇员名、最先出现的雇员名、表中的总行数和不同姓的人数。注意，这里有两个雇员的姓是“Brown”。

```
MAX(LAST_NAME)      MIN(LAST_NAME)      COUNT(*) COUNT(DISTINCTLAST_NAME)
------------------- ------------------- -------- ------------------------
Washington          Bell                      25                       24
```

5.7 写一个 SQL 查询，显示这章的 STATS 表中奇数列和偶数列的平均值，用同一函数，将空值按 0 计和忽略空值进行比较。

```
SELECT AVG(NVL(odd, 0)) "AVG ODDS", AVG(odd) 'IGNORING NULLS",
     AVG(NVL(even,0)) "AVG EVENS", AVG(even) "IGNORING NULLS"
FROM stats;
AVG ODDS   IGNORING NULLS AVG EVENS IGNORING NULLS
---------- -------------- --------- --------------
       5.5      7.3333333         7      9.3333333
```

通常这些函数忽略空值。可以用第 4 章的函数 NVL（）对分组函数施加影响，让其考虑所有的值，空值用另外的值来代替。

5.8 写一个 SQL 查询，显示每种工作职务人的平均工资值，总裁和副总裁除外。职务按字母表的顺序显示。

```
SELECT title, COUNT (*) "NUMBER", AVG(salary) "AVERAGE SALARY"
FROM s_emp
WHERE title <> 'President' AND NOT title LIKE 'VP%'
GROUP BY title
ORDER BY title;
TITLE                          NUMBER AVERAGE SALARY
------------------------------ ------ --------------
Sales Representative                4        1463.75
Stock Clerk                        11       945.45455
Warehouse Manager                   5          1591.4
```

在这个例子中，用 WHERE 子句来限制 GROUP BY 分组函数检查的行数。之所以使用 WHERE 子句，是因为要检查每一个单个的值。使用 ORDER BY 时，要将其放在 GROUP BY 和 HAVING 子句的后面。

5.9 写一个 SQL 查询，显示每种经理 ID 的平均工资，要求有该经理 ID 的人数要大于 1

```
SELECT manager_id, COUNT (*) "NUMBER",
AVG(salary) "AVERAGE SALARY"
FROM s_emp
GROUP BY manager_id
HAVING COUNT(*) > 1;
MAN     NUMBER  AVERAGE SALARY
--- ---------- --------------
1            4         2462.5
2            5         1591.4
3            4        1463.75
6            2           1170
7            3       951.66667
8            2            975
9            3       798.33333
```

5.10 写一个 SQL 查询，显示每种经理 ID 的平均工资，要求有该经理 ID 的人数要大于 1，而且平均工资大于 1200 美元。(提示：利用 5.9 的查询并使用 HAVING 子句进一步限制显示的行。)

```
SELECT manager_id, COUNT (*) "NUMBER",
     AVG(salary) "AVERAGE SALARY"
FROM s_emp
GROUP BY manager_id
HAVING COUNT(*) > 1 AND AVG(salary) > 1200;
MAN     NUMBER  AVERAGE SALARY
--- ---------- --------------
1            4         2462.5
2            5         1591.4
3            4        1463.75
```

5.11 写一个 SQL 查询，按部门分组，显示每种经理 ID 的平均工资，要求有该经理 ID 的

人数要大于 1。在每个部门中，再按经理 ID 分组

```
SELECT dept_id, manager_id, COUNT (*) "NUMBER",
       AVG(salary) "AVERAGE SALARY"
FROM s_emp
GROUP BY dept_id, manager_id
HAVING COUNT(*) > 1;
DEP MAN     NUMBER AVERAGE SALARY
--- --- --------- --------------
41    6         2           1170
42    7         2          997.5
44    9         2            800
```

5.12 写一个 SQL 查询，显示每种工作职务的最高工资、最低工资，其中要有 1 个以上的人拥有该职务。

```
SELECT title, COUNT (*) "NUMBER", MAX(salary) "MAX SALARY",
     MIN(salary) "MIN SALARY"
FROM s_emp
GROUP BY title
HAVING COUNT(*) > 1;
TITLE                      NUMBER MAX SALARY MIN SALARY
-------------------------- ------ ---------- ----------
Sales Representative            4       1515       1400
Stock Clerk                    11       1400        795
Warehouse Manager               5       1700       1500
```

5.13 写一个 SQL 查询，显示每个部门员工的最高工资、最低工资，其中该部门中的人数要在 1 个以上大于 1。按部门排序。

```
SELECT dept_id, MAX(salary) "MAX SALARY",
     MIN(salary) "MIN SALARY"
FROM s_emp
GROUP BY dept_id
HAVING COUNT(*) > 1
ORDER BY dept_id;
DEP MAX SALARY  MIN SALARY
--- ---------- ----------
31         2400       1400
41         2450        940
42         1650        795
43         1500        850
44         1700        800
45         1507        860
50         4500       2550
```

5.14 写一个 SQL 查询，显示部门名称和 ID 以及该部门的员工人数，要求部门的人数要大于 1。按部门员工总数降序显示。（提示：这个查询需要的信息，要从两个表中得到。）

```
SELECT s_dept.name "DEPARTMENT", s_emp.dept_id "ID",
     COUNT(s_emp.id) "EMPLOYEES"
FROM s_emp, s_dept
WHERE s_emp.dept_id = s_dept.id
GROUP BY s_emp.dept_id, s_dept.name
HAVING COUNT(s_emp.id) > 1
ORDER BY COUNT(s_emp.id) DESC;
DEPARTMENT            ID  EMPLOYEES
-------------------- --- ---------
Operations            41          4
Operations            45          4
Operations            42          3
Operations            44          3
Sales                 31          2
Operations            43          2
Administration        50          2
```

5.15 写一个SQL查询，显示部门名称和部门所在的地区名以及该部门的员工人数，要求部门的人数大于1。这个问题除了需要查询三个不同的表以外，与上一题类似。

```
SELECT s_dept.name "DEPARTMENT", s_region.name "REGION",
     COUNT(s_dept.id) "EMPLOYEES"
FROM s_dept, s_region, s_emp
WHERE s_dept.region_id = s_region.id
AND s_emp.dept_id = s_dept.id
GROUP BY s_dept.name, s_region.name
HAVING COUNT(s_dept.id) > 1;
DEPARTMENT           REGION                        EMPLOYEES
-------------------- ------------------------- ---------
Administration       North America                         2
Operations           Africa / Middle East                  2
Operations           Asia                                  3
Operations           Europe                                4
Operations           North America                         4
Operations           South America                         3
Sales                North America                         2
```

5.16 写一个SQL查询，显示地区ID和地区名以及该地区的客户数。

```
SELECT s_region.id, s_region.name,
     COUNT(s_customer.id) "CUSTOMERS"
FROM s_region, s_customer
WHERE s_region.id = s_customer.region_id
GROUP BY s_region.id, s_region.name;
ID  NAME                      CUSTOMERS
--- ------------------------- ---------
1   North America                    13
2   South America                     3
```

```
3   Africa / Middle East              2
4   Asia                              2
5   Europe                            4
6   Central America /Caribbean        1
```

补 充 题

注意：回答这些问题以前，重新运行 SG 脚本以刷新数据库。在所有这些问题中，我们使用定单表 S_ ODRD

```
CUS DATE_ORDE DATE_SHIP SAL     TOTAL PAYMEN
--- --------- --------- --- --------- ------
100 31-AUG-92 10-SEP-92 11     601100 CREDIT
101 31-AUG-92 15-SEP-92 14     8056.6 CREDIT
102 01-SEP-92 08-SEP-92 15       8335 CREDIT
103 02-SEP-92 22-SEP-92 15        377 CASH
104 03-SEP-92 23-SEP-92 15      32430 CREDIT
105 04-SEP-92 18-SEP-92 11    2722.24 CREDIT
106 07-SEP-92 15-SEP-92 12      15634 CREDIT
107 07-SEP-92 21-SEP-92 15     142171 CREDIT
108 07-SEP-92 10-SEP-92 13     149570 CREDIT
109 08-SEP-92 28-SEP-92 11    1020935 CREDIT
110 09-SEP-92 21-SEP-92 11    1539.13 CASH
111 09-SEP-92 21-SEP-92 11       2770 CASH
97  28-AUG-92 17-SEP-92 12      84000 CREDIT
98  31-AUG-92 10-SEP-92 14        595 CASH
99  31-AUG-92 18-SEP-92 14       7707 CREDIT
112 31-AUG-92 10-SEP-92 12        550 CREDIT
```

5.17 写一个 SQL 查询，显示所有支付方式是赊销的定单的总和及平均值。

5.18 写一个 SQL 查询，显示按支付方式分组的各支付方式的定单的总和及平均值。

5.19 写一个 SQL 查询，它将回答下面的问题：在定单表中，出现了多少个不同的订货日期，并与定单的总数相比较。

5.20 写一个 SQL 查询，它将回答下面的问题：在定单表中有多少客户和销售代理，并与定单的总数进行比较。

5.21 写一个 SQL 查询，显示所有定单中付款的最小值和最大值。

5.22 写一个 SQL 查询，回答如下的问题：每个销售代理定单的销售金额的平均值是多少？

5.23 写一个 SQL 查询，显示每个订货日期及该日期有多少定单。

5.24 写一个 SQL 查询，显示每个销售代表的不同支付方式的定单数、销售金额的平均值。

5.25 写一个 SQL 查询，显示定单中每个订货日期的金额最大值和最小值，要求每个订货日期有一个以上的定单。

5.26 写一个SQL查询，显示定单中每个订货日期的金额平均值，要求每个订货日期有一个以上的定单发出，而且定单的平均值大于1000美元。显示时要按平均值排序

5.27 写一个SQL查询，显示在定单中每个顾客的姓名及其定单数，要求每个顾客有一个以上的定单。显示时，按字母表的顺序显示顾客的姓名。

补充题答案

5.17

```
SELECT SUM(total) "SUM", AVG(total) "AVG"
FROM s_ord
WHERE payment_type = 'CREDIT';
       SUM        AVG
---------- ---------
2073210.8 172767.57
```

5.18

```
SELECT payment_type, COUNT(*) "NUMBER",
       SUM(total) "SUM", AVG(total) "AVG"
FROM s_ord
GROUP BY payment_type;
PAYMEN     NUMBER        SUM        AVG
------ ---------- ---------- ---------
CASH            4    5281.13 1320.2825
CREDIT         12 2073210.8 172767.57
```

5.19

```
SELECT COUNT(DISTINCT date_ordered) "NUM ORDER DATES",
     COUNT(*) "TOTAL"
FROM s_ord;
NUM ORDER DATES      TOTAL
--------------- ---------
              9         16
```

5.20

```
SELECT COUNT(DISTINCT customer_id) "CUSTOMERS",
     COUNT(DISTINCT sales_rep_id) "SALES REPS",
     COUNT(*) "TOTAL"
FROM s_ord;
CUSTOMERS SALES REPS      TOTAL
--------- ---------- ---------
       13          4         16
```

5.21

```
SELECT MAX(total) "HIGHEST", MIN(total) "LOWEST"
FROM s_ord;
  HIGHEST    LOWEST
--------- ---------
  1020935       377
```

5.22

```
SELECT sales_rep_id, AVG(total) "AVERAGE ORDER"
FROM s_ord
GROUP BY sales_rep_id;
SAL AVERAGE ORDER
--- -------------
11       271573.9
12       27129.75
13          91000
14        39632.4
```

5.23

```
SELECT date_ordered, COUNT(*) "NUMBER"
FROM s_ord
GROUP BY date_ordered;
DATE_ORDE     NUMBER
--------- ---------
28-AUG-92          1
31-AUG-92          5
01-SEP-92          1
02-SEP-92          1
03-SEP-92          1
04-SEP-92          1
07-SEP-92          3
08-SEP-92          1
09-SEP-92          2
```

5.24

```
SELECT sales_rep_id, payment_type, COUNT(*) "NUMBER",
     AVG(total) "AVERAGE"
FROM s_crd
GROUP BY sales_rep_id, payment_type;
SAL PAYMEN    NUMBER   AVERAGE
--- ------ --------- ---------
11  CASH           3 1562.0433
11  CREDIT         3 541585.75
12  CREDIT         4  27129.75
13  CREDIT         2.    91000
14  CASH           1       595
14  CREDIT         3 52644.867
```

5.25

```
SELECT date_ordered, MAX(total) "MAX ORDER",
     MIN(total) "MIN ORDER"
FROM s_ord
GROUP BY date_ordered
HAVING COUNT(*) > 1;
DATE_ORDE MAX ORDER MIN ORDER
--------- --------- ---------
31-AUG-92    601100       550
07-SEP-92    149570     15634
09-SEP-92      2770   1539.13
```

5.26

```
SELECT date_ordered, AVG(total) "AVERAGE ORDER"
FROM s_ord
GROUP BY date_ordered
HAVING COUNT(*) > 1 AND AVG(total) > 1000
ORDER BY AVG(total);
DATE_ORDE AVERAGE ORDER
--------- -------------
09-SEP-92       2154.565
07-SEP-92      102458.33
31-AUG-92      123601.72
```

5.27

```
SELECT s_customer.name, COUNT(s_ord.id) "ORDERS"
FROM s_customer, s_ord
WHERE s_customer.id = s_ord.customer_id
GROUP BY s_customer.name
HAVING COUNT(s_ord.id) > 1
ORDER BY s_customer.name;
NAME                           ORDERS
-------------- ----- ---------
Futbol Sonora                       2
Ladysport                           2
Muench Sports                       2
```

第 6 章　日期和时间信息的处理

6.1　日期和时间信息处理概述

在数据库中，最重要的工作之一是跟踪日期，不管是出生日期、放假日期、付款日期、定货日期还是装运日期等。数据库表中的数据也许是跨世纪的。另外，国际贸易必须处理各种时区问题。时间很重要，发运货物的时间可能是至关重要的。当显示查询结果时，日期和时间的格式化在查询或查询报告的可读性上会有很大差别。SQL 提供了丰富的、多样化的方法来处理日期和时间。除了能对日期做算术运算外，SQL 还提供了若干日期函数和格式化的实用工具程序。

本章所用的样本表是一家公司 2000 年的发薪日期表。按规定每月的 16 日和 30 日支付两次　假如遇到星期六或星期日，支票的日期将是在指定支付日期之前的星期五。表 PAY _ PERIODS 如表 6-1 所示。在这个表中有两列：每月第 1 张支票日期和每月第 2 张支票日期。

表 6-1　PAY　PERIODS 表

FIRST_CHE	SECOND_CH
14-JAN-00	28-JAN-00
16-FEB-00	29-FEB-00
16-MAR-00	30-MAR-00
14-APR-00	28-APR-00
16-MAY-00	30-MAY-00
16-JUN-00	30-JUN-00
14-JUL-00	28-JUL-00
16-AUG-00	30-AUG-00
15-SEP-00	29-SEP-00
16-OCT-00	30-OCT-00
16-NOV-00	30-NOV-00
15-DEC-00	29-DEC-00

6.2　日期的算术运算

第 4 章解释的算术运算符在 SQL 中是有效的。对日期有效的运算符是加号（+）和减

号（-），用这些运算符可以做三件事：

- 一个日期加一个天数的间隔，结果是新的日期。
- 从一个日期中减去一个天数的间隔，结果是新的日期。
- 两个日期相减，结果是两个日期之间间隔的天数。

在前一章中讲述了对某些日期数据的操作。在问题与答案 4.5 中有从一个日期减去另一日期的示例说明。只要用真正的日期，就可以使用第 1 章的比较运算符（>、<、>=、<=、=、<>）和第 3 章的布尔运算符（AND、OR、NOT）对日期数据进行比较。要比较以文字表示的日期，首先需要用 TO_DATE() 函数，正如将在 6.4 节中所示的那样，要将其转化成日期类型数据。问题与答案 4.27 说明了比较运算符在检验日期上的应用。

当操作的日期仅集中在天时，要十分小心。因为时间是以时、分和秒的方式存储的，算术运算可能会返回意想不到的值或者是天的小数（参见 6.4 节，要用ROUND() 或TRUNC() 函数解决这问题）。这里还将讨论其他算术运算的例子。

例 6.1

写一个 SQL 查询，列出支票之间的天数，然后加 7 天到每月的第 1 张支票日期上，而每月的第 2 张支票日期上，则减 7 天

```
SELECT second_check - first_check "BETWEEN",
     first_check + 7 "AFTER", second_check - 7 "BEFORE"
FROM pay_periods;
  BETWEEN AFTER     BEFORE
---------- --------- ---------
        14 21-JAN-00 21-JAN-00
        13 23-FEB-00 22-FEB-00
        14 23-MAR-00 23-MAR-00
        14 21-APR-00 21-APR-00
        14 23-MAY-00 23-MAY-00
        14 23-JUN-00 23-JUN-00
        14 21-JUL-00 21-JUL-00
        14 23-AUG-00 23-AUG-00
        14 22-SEP-00 22-SEP-00
        14 23-OCT-00 23-OCT-00
        14 23-NOV-00 23-NOV-00
        14 22-DEC-00 22-DEC-00
```

在查询结果表上，除发薪在 2 月份的 29 日外，显示的日期都是相同的。因为发薪间隔总是 14 天。

例 6.2

写一个 SQL 查询，列出从 1 月到 6 月的发薪日期。

```
SELECT first_check, second_check
FROM pay_periods
WHERE first_check > '01-JAN-00' AND first_check < '01-JUL-00';
FIRST_CHE SECOND_CH
--------- ---------
14-JAN-00 28-JAN-00
16-FEB-00 29-FEB-00
16-MAR-00 30-MAR-00
14-APR-00 28-APR-00
16-MAY-00 30-MAY-00
16-JUN-00 30-JUN-00
```

注意，在这个查询中，把 1 月 1 日和 7 月 1 日作为条件判断的边界。如果 WHERE 子句的第 2 个比较运算用 6 月的最后一天的话，也会出现相同的结果。

6.3 日期函数

若干内部函数对大多数重要的日期处理是有用的。这些函数列在表 6-2 中。其中大多数函数将在下面两节中以实例说明。NEW _TIME()、GREATEST() 和LEAST() 将在 6.4 节中解释，一起进行讨论的还有日期和时间的格式化问题

表 6-2 日期函数

函数名	输入参数	返回值
ADD_MONTHS(d, n)	d = 日期 n = 月数	日期 d 加 n 个月
LAST_DAY(d)	d = 日期	包含 d 的月份的最后一天的日期
MONTH_BETWEEN(d, e)	d = 日期 e = 日期	日期 d 与 e 之间的月份数，e 先于 d
NEW_TIME(d, a, b)	d = 日期 a = 时区（字符） b = 时区（字符）	a 时区的日期和时间 d 在 b 时区的日期和时间
NEXT_DAY(d, day)	d = 日期 day = 星期	比日期 d 晚，由 day 指定的周几的日期
SYSDATE	无	当前的系统日期和时间
GREATEST(d1, d2, .. dn)	d1, ...dn = 日期列表	给出的日期列表中最后的日期
LEAST(d1, d2, .. dn)	d1, ...dn = 日期列表	给出的日期列表中最早的日期

6.3.1 SYSDATE 函数

内部函数 SYSDATE 从正运行 RDBMS 的特定计算机的系统时钟中返回当前的日期和时间。该函数的名称对于不同的 RDBMS 可能不同，经常用 TODAY 和 CURRENT DATE 两

个函数名，不管函数的名称是什么，它们的规则都是相同的。

例 6.3

```
用 DUAL 表，列出当前日期.
SELECT SYSDATE
FROM DUAL;
SYSDATE
---------
23-JAN-00
```

假如今天是 2000 年 1 月 23 日，就会出现上面的显示结果。这个命令总是显示当前日期。

6.3.2　DAY 和 MONTH 函数

SQL 提供了若干个不同的处理月份的函数。用 ADD_MONTHS(d, n) 函数能把若干个月份数 n 加到日期 d 中。同样用这个函数，也能做月份的相减，只不过 n 为负数。MONTH_BETWEEN(d, e) 函数将显示两个日期之间的月份数。LAST_DAY(d) 给出由 d 指定的月的最后一天的日期，因为每月的天数是变化的，所以这个函数是必要的，对于 2 月尤其有帮助。这些函数在使用时，可以任意地组合，正如下面的例子所示。

例 6.4

显示 2 个月前的这一天的日期，以及 2 个月后的这一天的日期，并显示它们之间有几个月。

```
SELECT SYSDATE, ADD_MONTHS(SYSDATE, -2) "2 MO. AGO",
       ADD_MONTHS(SYSDATE, 2) "2 MO. FROM NOW",
       MONTHS_BETWEEN(ADD_MONTHS(SYSDATE, 2),
       ADD_MONTHS(SYSDATE, -2)) "BETWEEN"
FROM DUAL;
SYSDATE   2 MO. AGO 2 MO. FRO   BETWEEN
--------- --------- --------- ---------
23-JAN-00 23-NOV-99 23-MAR-00         4
```

注意，SQL 的这个版本是支持 2000 年的。假如当前日期是 2000 年 1 月 23 日，系统知道 1999 年先于 2000 年。因此，必须保证任何一个新日期输进数据库时，使用 4 位数字。

例 6.5

按照公司例子的说明，第 2 个发薪日期是 30 号或 30 号前的星期五，不是该月的最后一天。显示第 2 个发薪日期以及 2000 年 1 月到 6 月每月的最后一天的日期，然后显示两者之

间的天数

```
SELECT second_check, LAST_DAY(second_check),
     LAST_DAY(second_check) - second_check "BETWEEN"
FROM pay_periods
WHERE second_check > '01-JAN-00' AND second_check < '01-JUL-00';
SECOND_CH LAST_DAY(    BETWEEN
--------- --------- ---------
28-JAN-00 31-JAN-00         3
29-FEB-00 29-FEB-00         0
30-MAR-00 31-MAR-00         1
28-APR-00 30-APR-00         2
30-MAY-00 31-MAY-00         1
30-JUN-00 30-JUN-00         0
```

2000年是闰年，前面例子的结果表反映了这个事实。记住，为了找到日期之间的月份数，需要用MONTHS_BETWEEN()函数。可以通过将两个日期简单相减得到两个日期之间的天数。

另外一个非常有用的函数是NEXT_DAY(d, day)，用它可找到一周中指定的一天的日期。指定一周中希望的一天，写法上有要求（例如，“Monday”、“Tuesday”等），则函数返回值是跟在d后那天的日期。例如，我们总是想在星期五或星期一付款，这就需要用此函数找到付款日期，如下例所示

例6.6

假如总是要在第1张支票后的星期五发薪，显示这些日期。

```
SELECT first_check, NEXT_DAY(first_check,'Friday') "FRIDAY"
FROM pay_periods;
FIRST_CHE FRIDAY
--------- ---------
14-JAN-00 21-JAN-00
16-FEB-00 18-FEB-00
16-MAR-00 17-MAR-00
14-APR-00 21-APR-00
16-MAY-00 19-MAY-00
16-JUN-00 23-JUN-00
14-JUL-00 21-JUL-00
16-AUG-00 18-AUG-00
15-SEP-00 22-SEP-00
16-OCT-00 20-OCT-00
16-NOV-00 17-NOV-00
15-DEC-00 22-DEC-00
```

6.4　日期和时间的格式化

要以某种格式显示日期，必须将日期转变成字符串。第 4 章用实例说明了使用 TO_NUMBER() 和 TO_CHAR() 函数能从字符转变成数值和从数值转变成字符。与此类似的是，用 TO_CHAR() 和 TO_DATE() 函数能将日期转变成字符和从字符转变成日期。用这些内部函数的一种形式舍入和截断日期是可能的。在表 6-3 中，列出了能用于格式化日期的函数。

表 6-3　日期格式化函数

函数名	输入参数	返回值
TO_CHAR(d [, fmt])	d = 日期值	日期 d 按 fmt 指定的格式转变成字符串
	fmt = 字符串格式	fmt 缺省值的宽度正好与有效数字一样
TO_DATE(st [, fmt])	st = 字符串值	字符串 st 按 fmt 指定的格式转成日期值
	fmt = 日期格式	若 fmt 忽略，st 要用缺省格式
ROUND(d [, fmt])	d = 日期值	日期 d 按 fmt 指定格式舍入到最近的日期
	fmt = 字符串格式	
TRUNC(d [, fmt])	d = 日期值	日期 d 按 fmt 指定格式截断到最近的日期
	fmt = 字符串格式	

6.4.1　TO_DATE()和TO_CHAR()函数及格式化

TO_CHAR(d [, fmt]) 和 TO_DATE(st [, fmt]) 函数用来对指定数据在字符串和日期之间来回转换。无论什么时候，想打印漂亮格式的日期，都必须转换成字符串。假如要比较以文字表示的日期值，也必须先将其转换成实际的日期。在这两种情况下，使用格式化参数 fmt，以按期望的格式显示日期和时间。最常用的日期格式如表 6-4 所示。如参数 fmt 省略，则使用缺省格式 'DD-MMM-YY'，其中 DD、YY 是 2 位数字，分别代表日和年，MMM 是三个英文大写字母，是月份的缩写。中间是连字符，最后用单引号括起来（例如，'01-JAN-00'，'18-FEB-97'）。

表 6-4　日期格式

格式代码	说明	举例或可取值的范围
DD	该月的某一天	1 - 31
DY	三个英文大写字母的缩写，表示这一天是周几	SUN，...SAT
DAY	该天完整的大写名称，填满 9 个字符	SUNDAY，...SATURDAY
MM	月份数	1 - 12
MON	三个大写英文字母缩写的月份名	JAN，...DEC

（续表）

格式代码	说明	举例或可取值的范围
MONTH	大写英文的月份名，填满9个字符	JANUARY，...DECEMBER
RM	月份的罗马数字	Ⅰ，...Ⅻ
YY或YYYY	两位数字的年或四位数字的年	00，99或例如1987，2002，1776
HH:MI:SS	时：分：秒	例如12:45:58
HH12或HH24	以12小时或24小时的格式显示	1－12或1－24
MI	小时的分钟数	0－59
SS	分钟的秒数	0－59
AM或PM	上下午指示符	AM或PM
SP	后缀SP要求拼写出任何数值字段（如YY，MM，DD等）	例如：ONE，NINETEEN，TWO THOUSAND TWO
TH	后缀TH表示添加的数字是序数	例如：4th，1st
FM	前缀对月或日或年值，禁止填充	MONDAY尾部不扩充空格

许多标点符号可以合并到格式串中，它们将复制到日期的显示中。例如，或许有人想用像“January 23，2000”这样的格式来代替缺省格式显示日期。这样，日期字段将会被转换成字符串来完成这个任务。

例6.7

按从1月到6月的顺序，显示每月第1次发薪的日期，格式为“Month ddth，yyyy”。

a.

初次尝试如下：

```
SELECT TO_CHAR(first_check, 'MONTH DD, YYYY')
FROM pay_periods
WHERE first_check > '01-JAN-00' AND first_check < '01-JUL-00';
TO_CHAR(FIRST_CHECK,'MONTHDD,YYYY')
---------------- ------------------
JANUARY   14, 2000
FEBRUARY  16, 2000
MARCH     16, 2000
APRIL     14, 2000
MAY       16, 2000
JUNE      16, 2000
```

b.

上面显示的结果图表中，每月的名称填充到9个字符。第2次用FM禁止填充日期，以使其看起来更自然些。还有，使用TH标明日期用序数词。

```
SELECT TO_CHAR(first_check, 'FMMONTH DDTH, YYYY ')
FROM pay_periods
WHERE first_check > '01-JAN-00' AND first_check < '01-JUL-00';
TO_CHAR(FIRST_CHECK,'FMMONTHDDTH,YYYY )
---------- ---------- --------- ----------
JANUARY 14TH, 2000
FEBRUARY 16TH, 2000
MARCH 16TH, 2000
APRIL 14TH, 2000
MAY 16TH, 2000
JUNE 16TH, 2000
```

正如表6-4所示，月份能显示成数字、名称、3个字母的缩写和罗马数字。例6.8表示了显示每个月的多种方式。

例6.8

用单词、缩写单词、数字和罗马数显示一年的月份。

```
SELECT TO_CHAR(first_check,'MONTH') "WORDS",
       TO_CHAR(first_check, 'MON') "ABBREVIATED",
       TO_CHAR(first_check, 'MM') "NUMBERS",
       TO_CHAR(first_check, 'RM') "ROMAN NUMERALS"
FROM pay_periods;
WORDS        ABBREVIATED  NUMBERS  ROMAN NUMERALS
-----------  -----------  -------  ---------------
JANUARY      JAN          01       I
FEBRUARY     FEB          02       II
MARCH        MAR          03       III
APRIL        APR          04       IV
MAY          MAY          05       V
JUNE         JUN          06       VI
JULY         JUL          07       VII
AUGUST       AUG          08       VIII
SEPTEMBER    SEP          09       IX
OCTOBER      OCT          10       X
NOVEMBER     NOV          11       XI
DECEMBER     DEC          12       XII
```

日期也有多种显示方式。后缀SP加到任一数上，表示拼写出这个数字。后缀TH加到任一数上，表示用序数词。例6.9说明了用这个方式显示的2000年1月第一次发薪的日期。

例6.9

显示1月份第一次发薪的日期。用单词、序数、数字拼写和序数拼写多种方式显示。

```
SELECT TO_CHAR(first_check, 'DAY') "WORDS",
       TO_CHAR(first_check, 'DDTH') "ORDINAL",
       TO_CHAR(first_check, 'DDSP') "SPELLED OUT",
       TO_CHAR(first_check, 'DDSPTH') "ORD SPELLED OUT"
FROM pay_periods
WHERE first_check =  14-JAN-00';
WORDS      ORDINAL  SPELLED OUT     ORD SPELLED OUT
---------- -------- --------------- -----------------------
FRIDAY     14TH     FOURTEEN        FOURTEENTH
```

6.4.2 ROUND()和TRUNC()函数

ROUND(d [, fmt]) 和TRUNC(d [, fmt]) 函数允许舍入或者截断时间项的任何一部分。当日期以缺省方式输入时，如‘02-JAN-00’，系统假定的缺省时间是午夜或一天的开始。SYSDATE的值总是包含有当前日期和时间。因此，有关日期的计算有时会产生带小数的天数。见例6.10，为解决这个问题，用ROUND() 和TRUNC() 函数。ROUND()函数总是返回最接近午夜的时间，或者是当前日期的或者是下一天的，这要由它在午前还是午后来决定。另一方面，TRUNC() 函数总是将时间设置成当前日期的午夜。

例 6.10

列出当前日期（假定是2000年1月27日）和2月份第一个发薪日之间的天数。

a.

```
SELECT SYSDATE "CURRENT",
    ABS(SYSDATE - first_check) "UNTIL FIRST"
FROM pay_periods
WHERE first_check = '16-FEB-00';
CURRENT   UNTIL FIRST
--------- ------------
27-JAN-00   19.516921
```

注意，ABS函数返回的是带小数的天数。这是因为当前时间是用SYSDATE函数得到的。为了纠正这一点，用ROUND() 和TRUNC() 函数。假如在午前，它们将返回相同的天数。b部分显示了在午前查询的结果。c部分显示的是午后查询的结果。午后查询的情况下，舍入时间使发薪的天数少了。

b.

```
SELECT SYSDATE "CURRENT",
       TO_CHAR(SYSDATE, 'HH:MI:SS AM') "TIME",
       ABS(ROUND(SYSDATE) - first_check) "UNTIL FIRST",
       ABS(TRUNC(SYSDATE) - first_check) "UNTIL FIRST"
FROM pay_periods
```

```
WHERE first_check = '16-FEB-00';
CURRENT  TIME                          UNTIL FIRST UNTIL FIRST
-------- ----------------------------- ----------- -----------
27-JAN-00 11:40:03 AM                           20          20
```

c.

```
SELECT SYSDATE "CURRENT",
    TO_CHAR(SYSDATE, 'HH:MI:SS AM') "TIME",
    ABS(ROUND(SYSDATE) - first_check) "UNTIL FIRST",
    ABS(TRUNC(SYSDATE) - first_check) "UNTIL FIRST"
FROM pay_periods
WHERE first_check = '16-FEB-00';
CURRENT   TIME                         UNTIL FIRST UNTIL FIRST
--------- ---------------------------- ----------- -----------
27-JAN-00 12:03:04 PM                           19          20
```

6.4.3 GREATEST()和LEAST()函数

GREATEST(d1, d2, ... dn) 和LEAST(d1, d2, ... dn) 函数分别从给出的日期列表 (d1, d2, ... dn) 中，挑选出最晚的和最早的日期。函数的日期参数个数不限，可检验任意多个日期。然而，要确信被检验的是日期而不是字符串，这一点非常重要。

例 6.11

从1月到6月，显示每月最早的和最晚的支票日期

```
SELECT LEAST(second_check, first_check) 'EARLIER",
GREATEST(second_check, first_check) "LATER"
FROM pay_periods
WHERE second_check > '01-JAN-00' AND second_check < '01-JUL-00';
EARLIER   LATER
--------- ----- ---
14-JAN-00 28-JAN-00
16-FEB-00 29-FEB-00
16-MAR-00 30-MAR-00
14-APR-00 28-APR-00
16-MAY-00 30-MAY-00
16-JUN-00 30-JUN-00
```

上面的查询有效地发挥了作用，这是因为两个字段都是日期型数据。当日期参数用文字值时，就没有这么简单了，如下例所示。

例 6.12

从以下三个日期值：January 1, 1999、November 12, 1998 和 October 19, 1997 中，显

示最近的日期和最早的日期。

a.

该查询的首次尝试如下，请仔细检查输出：

```
SELECT GREATEST('01-Jan-99', '12-Nov-98','19-Oct-97 ) "LATEST",
     LEAST('01-Jan-99', '12-Nov-98','19-Oct-97 ) "EARLIEST'
FROM DUAL;
LATEST    EARLIEST
--------- ---------
19-Oct-97 01-Jan-99
```

很清楚，系统对待这些选择，是按字符串而不是按日期来处理的。因此，以“01”开始的一个先来，以“09”开始的一个后到。要比较日期必须使用 TO_DATE() 函数先将其转换成日期型数据。

b.

第二次尝试纠正了查询的版本，如下所示。

```
SELECT GREATEST(TO_DATE('01-Jan-99'), TO_DATE('12-Nov-98 ),
     TO_DATE( 19-Oct-97')) "LATEST",
     LEAST(TO_DATE('01-Jan-99'), TO_DATE('12-Nov-98'),
     TO_DATE('19-Oct-97')) "EARLIEST"
FROM DUAL;
LATEST    EARLIEST
--------- ---------
01-JAN-99 19-OCT-97
```

注意，现在的函数参数是按日期型数据对待的，结果是正确的。在你的 SQL 版本中，这些函数不一定能处理 2000 年问题，在使用它们之前，进行测试总是会有帮助的。

6.4.4 NEW_TIME()函数

NEW_TIME(d, a, b) 函数，返回 a 时区的日期和时间 d 在 b 时区对应的日期和时间。这个函数目前并不支持所有的时区。表 6-5 展示了对格林威治、英格兰和夏威夷之间时区的支持。用你的当前版本，检查 SQL 文档所支持的时区列表。例 6.13 说明了如何使用这些函数。

表 6-5 时区

时区	解释
AST/ADT	大西洋标准时/白昼时间
BST/BDT	白令时间标准/白昼时间
CST/CDT	中部标准时/白昼时间
EST/EDT	东方标准时/白昼时间
GMT	格林威治时间

（续表）

时区	解释
HST/HDT	阿拉斯加-夏威夷标准时/白昼时间
MST/MDT	山地标准时/白昼时间
NST	纽分兰标准时间
PST/PDT	太平洋标准时/白昼时间
YST/YDT	育空标准时/白昼时间

例 6.13

显示在东方时区、英格兰和夏威夷时区的当前时间。

```
SELECT TO_CHAR(SYSDATE, 'HH24:MI AM') "EASTERN",
TO_CHAR(NEW_TIME(SYSDATE, 'EST','GMT'), 'HH24:MI AM') "GREENWICH",
TO_CHAR(NEW_TIME(SYSDATE, 'EST','HST'), 'HH24:MI AM') "HAWAII"
FROM DUAL;
EASTERN            GREENWICH          HAWAII
------------------ ------------------ ------------------
15:23 PM           20:23 PM           10:23 AM
```

问题与答案

注意：解答这些问题前，重新运行 SG 脚本以刷新数据库。在这节的一些问题中，使用雇员表 S_EMP。

```
LAST_NAME       FIRST_NAME      TITLE                     START_DAT
--------------- --------------- ------------------------- ---------
Martin          Carmen          President                 03-MAR-90
Smith           Doris           VP, Operations            08-MAR-90
Norton          Michael         VP, Sales                 17-JUN-91
Quentin         Mark            VP, Finance               07-APR-90
Roper           Joseph          VP, Administration        04-MAR-90
Brown           Molly           Warehouse Manager         18-JAN-91
Hawkins         Roberta         Warehouse Manager         14-MAY-90
Burns           Ben             Warehouse Manager         07-APR-90
Catskill        Antoinette      Warehouse Manager         09-FEB-92
Jackson         Marta           Warehouse Manager         27-FEB-91
Henderson       Colin           Sales Representative      14-MAY-90
Gilson          Sam             Sales Representative      18-JAN-92
Sanders         Jason           Sales Representative      18-FEB-91
Dameron         Andre           Sales Representative      09-OCT-91
Hardwick        Elaine          Stock Clerk               07-FEB-92
Brown           George          Stock Clerk               08-MAR-90
```

```
Washington      Thomas          Stock Clerk          09-FEB-91
Patterson       Donald          Stock Clerk          06-AUG-91
Bell            Alexander       Stock Clerk          26-MAY-91
Gantos          Eddie           Stock Clerk          30-NOV-90
Stephenson      Blaine          Stock Clerk          17-MAR-91
Chester         Eddie           Stock Clerk          30-NOV-90
Pearl           Roger           Stock Clerk          17-OCT-90
Dancer          Bonnie          Stock Clerk          17-MAR-91
Schmitt         Sandra          Stock Clerk          09-MAY-91
```

6.1 用单词、序数、数字拼写和序数拼写方式，显示2000年1月1日的日期。

```
SELECT TO_CHAR(TO_DATE('01-JAN-00'), 'DAY') "WORDS",
     TO_CHAR (TO_DATE('01-JAN-00'), 'DDTH') "ORDINAL",
     TO_CHAR(TO_DATE('01-JAN-00'), 'DDSP') "SPELLED OUT",
     TO_CHAR(TO_DATE('01-JAN-00'), 'DDSPTH') "ORD SPELLED OUT"
FROM DUAL;
WORDS        ORDINAL  SPELLED OUT  ORD SPELLED OUT
------------ -------- ------------ --------------------
SATURDAY     01ST     ONE          FIRST
```

注意，当在DUAL表中用字面值时，必须首先将它们转换成日期，然后，再按格式转换成字符串。

6.2 假定新雇员1998年7月6日到公司工作。写一个SQL查询，显示复查日期，假定标准的复查日期是到公司后60天进行。假定你只在星期五做复查工作，找一个跟在复查日期后最近的星期五。注意，为找到正确的1998年历，创建日期时，要用4位数字表示年代。

```
SELECT TO_DATE('06-JUL-1998','DD-MON-YYYY') "HIRE DATE",
TO_DATE('06-JUL-1998','DD-MON-YYYY') + 60 "REVIEW DATE",
NEXT_DAY(TO_DATE('06-JUL-1998','DD-MON-YYYY') + 60, 'FRIDAY') "FRIDAY"
FROM DUAL;
HIRE DATE  REVIEW DA  FRIDAY
---------  ---------  ---------
06-JUL-98  04-SEP-98  11-SEP-98
```

6.3 一位雇员1990年5月14日到公司工作，另一位雇员1992年1月18日到公司工作。写一个SQL查询，显示两人到公司工作相差的月数及天数。

```
SELECT MONTHS_BETWEEN('18-JAN-92', '14-MAY-90') "MONTHS",
     TO_DATE('18-JAN-92') - TO_DATE('14-MAY-90') "TOTAL DAYS"
FROM DUAL;
   MONTHS  TOTAL DAYS
---------- ----------
20.129032         614
```

注意，MONTHS_BETWEEN()函数返回了小数。可以使用ROUND()函数避免这种情况。如下所示：ROUND(MONTHS_BETWEEN('18-JAN-92', '14-MAY-90'))。假如使用了ROUND()函数，月份总数应是20。

6.4　一个人今天到公司（例如，2000 年 1 月 27 日）工作。写一个 SQL 查询，显示两个月后复查的日期，展示 60 天后的日期并进行比较。

```
SELECT SYSDATE "TODAY", ADD_MONTHS(SYSDATE, 2) "2 MONTHS",
     SYSDATE + 60 '60 DAYS"
FROM DUAL;
TODAY      2 MONTHS   60 DAYS
---------- ---------- --------
27-JAN-00 27-MAR-00 27-MAR-00
```

6.5　试作 6.4 题相同的查询，假如复查是 6 个月或 180 天后，显示复查的日期。

```
SELECT SYSDATE "TODAY", ADD_MONTHS(SYSDATE, 6) "6 MONTHS",
     SYSDATE + 180 "180 DAYS"
FROM DUAL;
TODAY      6 MONTHS   180 DAYS
---------- ---------- ---------
27-JAN-00 27-JUL-00 25-JUL-00
```

6.6　写一个 SQL 查询，从 S_EMP 表中，显示雇佣的所有销售代表工作的开始日期和这个月的最后的日期。

```
SELECT last_name, start_date,
     LAST_DAY(start_date) "LAST DAY OF MONTH"
FROM s_emp
WHERE title = 'Sales Representative';
LAST_NAME              START_DAT LAST DAY
---------------------- --------- ---------
Henderson              14-MAY-90 31-MAY-90
Gilson                 18-JAN-92 31-JAN-92
Sanders                18-FEB-91 28-FEB-91
Dameron                09-OCT-91 31-OCT-91
```

6.7　写一个 SQL 查询，显示所有仓库经理到公司工作那周的星期一的日期。

```
SELECT last_name, start_date,
     NEXT_DAY(start_date, 'Monday') "MONDAY"
FROM s_emp
WHERE title = 'Warehouse Manager';
LAST_NAME              START_DAT MONDAY
---------------------- --------- ---------
Brown                  18-JAN-91 21-JAN-91
Hawkins                14-MAY-90 21-MAY-90
Burns                  07-APR-90 09-APR-90
Catskill               09-FEB-92 10-FEB-92
Jackson                27-FEB-91 04-MAR-91
```

6.8　为确定资历，需要知道每位副总裁从来到公司之日起到 1997 年 1 月 1 日之间的确切天数，还需要知道到今天（例如，2000 年 2 月 11 日）为止有多少天。写一个 SQL 查询，

显示这些信息。记住，当对以文字表示的日期值进行算术运算时，必须用TO_DATE()对其进行转换。

```
SELECT last_name, TO_DATE( '01-JAN-1997' ,'DD-MON-YYYY' )
     - start_date "START TO 1997",
     ROUND(SYSDATE) - start_date "START TO NOW"
FROM s_emp
WHERE title LIKE 'VP%';
LAST_NAME              START TO 1997  START TO NOW
---------------------- -------------- ------------
Smith                            2491         3628
Norton                           2025         3162
Quentin                          2461         3598
Roper                            2495         3632
```

6.9 写一个SQL查询，显示总裁工作的开始日期，以如下几种方式显示：缺省、月和年、序数日期和一周的日期。

```
SELECT start_date, TO_CHAR(start_date, 'MM/YY') "MM/YY",
     TO_CHAR(start_date, 'DDTH') "ORDINAL",
     TO_CHAR(start_date, 'DAY') "DAY"
FROM s_emp
WHERE title = 'President';
START_DAT MM/YY ORDINAL DAY
--------- ----- ------- ------------------------------
03-MAR-90 03/90 03RD    SATURDAY
```

6.10 写一个SQL查询，显示副总裁的姓名和开始工作的日期，把这些信息放在一个语句格式中，像“GEORGE JONES was hired 03rd of JUNE，1996”一样。

```
SELECT first_name || ' ' || last_name || ' was hired ' ||
     TO_CHAR(start_date,'DDTH') || ' of ' ||
     TO_CHAR(start_date,'FMMONTH') || ', ' ||
     TO_CHAR(start_date,'YYYY') "VP's"
FROM s_emp
WHERE title LIKE 'VP%';
VP's
-----------------------------------------------
Doris Smith was hired 08TH of MARCH, 1990
Michael Norton was hired 17TH of JUNE, 1991
Mark Quentin was hired 07TH of APRIL, 1990
Joseph Roper was hired 04TH of MARCH, 1990
```

6.11 写一个SQL查询，显示对销售代表到公司的日期、1992年1月1日和1991年1月1日三个日期的比较，显示最晚和最早的日期。

a.

初次尝试如下：

```
SELECT last_name,
GREATEST (start_date,
TO_DATE('01-JAN-92 '),TO_DATE('01-JAN-91')) "GREATEST",
LEAST (start_date,
TO_DATE('01-JAN-92'),TO_DATE( 01-JAN-91')) "LEAST"
FROM s_emp
WHERE title = 'Sales Representative';
LAST_NAME                 GREATEST  LEAST
------------------------- --------- ---------
Henderson                 01-JAN-92 14-MAY-90
Gilson                    01-JAN-92 18-JAN-92
Sanders                   01-JAN-92 18-FEB-91
Dameron                   01-JAN-92 09-OCT-91
```

仔细检查该表后可以发现此次查询是失败的。在它展示的查询结果中，18-JAN-92 早于 01-JAN-92。记住，在本世纪创建一个新日期时，应该用完整的 4 位数字格式。

b.

纠正上面的查询，查询结果的图表如下：

```
SELECT last_name,
GREATEST (start_date, TO_DATE('01-JAN-1992','DD-MON-YYYY'),
     TO_DATE('01-JAN-1991','DD-MON-YYYY')) "GREATEST",
     LEAST (start_date, TO_DATE( 01-JAN-1992','DD-MON-YYYY'),
     TO_DATE('01-JAN-1991','DD-MON-YYYY')) "LEAST"
FROM s_emp
WHERE title = 'Sales Representative';
LAST_NAME                 GREATEST  LEAST
------------------------- --------- ---------
Henderson                 01-JAN-92 14-MAY-90
Gilson                    18-JAN-92 01-JAN-91
Sanders                   01-JAN-92 01-JAN-91
Dameron                   01-JAN-92 01-JAN-91
```

6.12　写一个 SQL 查询，显示加利福尼亚和育空的当前时间。假定当前时间是在大西洋的标准时区。

```
SELECT TO_CHAR(SYSDATE,'HH24:MI PM') "ATLANTIC",
  TO_CHAR(NEW_TIME(SYSDATE,'AST','PST'), 'HH24:MI AM') "CALIFORNIA",
  TO_CHAR(NEW_TIME(SYSDATE,'AST','YST'), 'HH24:MI AM') "YUKON"
FROM DUAL;
ATLANTIC   CALIFORNIA   YUKON
---------- ------------ ------------------------------
15:22 PM   11:22 AM     10:22 AM
```

6.13　写一个 SQL 查询，显示每位仓库经理的姓名和在这里工作的月数。结果按月数排序，时间最大的排在最前面。当然，月数要舍入到最接近的整数。

```
SELECT last_name,
  ROUND(MONTHS_BETWEEN(SYSDATE, start_date)) "BETWEEN"
FROM s_emp
WHERE title = 'Warehouse Manager'
ORDER BY ROUND(MONTHS_BETWEEN(start_date, SYSDATE)) ASC;
```

假如SYSDATE是2000年1月27日，则查询的结果如下：

```
LAST_NAME                 BETWEEN
------------------------- -------
Burns                         118
Hawkins                       116
Brown                         108
Jackson                       107
Catskill                       96
```

6.14 查询每位库存管理员开始工作的日期和2000年1月1日之间的月数和天数。将答案放在一个语句的格式中，按工作的月数排序。

```
SELECT first_name || ' ' || last_name ||' has been employed '||
     TRUNC(MONTHS_BETWEEN(TO_DATE('01-JAN-2000','DD-MON-YYYY'),
      start_date))|| ' months or ' ||
     TO_CHAR(TO_DATE('01-JAN-2000', 'DD-MON-YYYY')
     - start_date) ||' days.' "EMP. HISTORY AS OF 1-1-2000"
FROM s_emp
WHERE title = 'Stock Clerk'
ORDER BY ROUND(MONTHS_BETWEEN(start_date,
     TO_DATE('01-JAN-2000', 'DD-MON-YYYY'))) DESC;
EMP. HISTORY AS OF 1-1-2000
--- ----------------------------------------------------------
Elaine Hardwick has been employed 94 months or 2885 days.
Donald Patterson has been employed 100 months or 3070 days.
Alexander Bell has been employed 103 months or 3142 days.
Sandra Schmitt has been employed 103 months or 3159 days.
Blaine Stephenson has been employed 105 months or 3212 days.
Bonnie Dancer has been employed 105 months or 3212 days.
Thomas Washington has been employed 106 months or 3248 days.
Eddie Gantos has been employed 109 months or 3319 days.
Eddie Chester has been employed 109 months or 3319 days.
Roger Pearl has been employed 110 months or 3363 days.
George Brown has been employed 117 months or 3586 days.
```

补 充 题

注意：重新运行SG脚本以刷新该数据库。在这节的所有问题中，使用定单表S_ORD。

```
CUS DATE_ORDE DATE_SHIP SAL     TOTAL PAYMEN
--- --------- --------- --- --------- ------
100 31-AUG-92 10-SEP-92 11     601100 CREDIT
101 31-AUG-92 15-SEP-92 14     8056.6 CREDIT
102 01-SEP-92 08-SEP-92 15       8335 CREDIT
103 02-SEP-92 22-SEP-92 15        377 CASH
104 03-SEP-92 23-SEP-92 15      32430 CREDIT
105 04-SEP-92 18-SEP-92 11    2722.24 CREDIT
106 07-SEP-92 15-SEP-92 12      15634 CREDIT
107 07-SEP-92 21-SEP-92 15     142171 CREDIT
108 07-SEP-92 10-SEP-92 13     149570 CREDIT
109 08-SEP-92 28-SEP-92 11    1020935 CREDIT
110 09-SEP-92 21-SEP-92 11    1539.13 CASH
111 09-SEP-92 21-SEP-92 11       2770 CASH
97  28-AUG-92 17-SEP-92 12      84000 CREDIT
98  31-AUG-92 10-SEP-92 14        595 CASH
99  31-AUG-92 18-SEP-92 14       7707 CREDIT
112 31-AUG-92 10-SEP-92 12        550 CREDIT
```

6.15 假定在8月31日收到的定单需要在21天后发货。写一个SQL查询，显示预定的发货日期

6.16 假定在8月31日收到的定单，需要在2个月后发货。写一个SQL查询，显示预定的发货日期

6.17 写一个SQL查询，显示8月份所有定单的定货日期和发货日期之间的天数。

6.18 写一个SQL查询，显示最完整的当前日期和时间的版本，包括时、分、秒以及AM或PM。用两种方式显示：12小时和24小时制。

6.19 写一个SQL查询，显示9月份定单的不同定货日期，用两种方式写出：带有数字和不带有数字的形式，如“4TH of 9TH month”和“fourth of September”。

6.20 写一个SQL查询，显示8月份定单的不同定货日期，写出两种方式：月和年（08/92）和像“31 August，1992”的格式。

6.21 写一个SQL的查询，显示当前月的最后一天、两个月前的最后一天以及下一个月的最后一天。

6.22 写一个SQL查询，打印在9月20日后每个不同的发货日期的下一个星期五的日期。当然，在WHERE子句中要用4位数字的日期。

6.23 写一个SQL查询，显示当前日期和2002年圣诞节之间的月数，如果月数是小数要舍入到最接近的月数。

6.24 写一个SQL查询，显示芝加哥和丹佛的当前时间。假定当前时间用格林威治时表示。

6.25 写一个SQL查询，将9月份的每个发货日期与1992年9月15日和1992年9月20日比较，显示最早的日期和最晚的日期。当然，要用4位数字的日期。

6.26 写一个SQL查询，将8月份的每一个定单显示成一个句子，如“The order from customer <customer_id> <date_ordered> was shipped on <date_shipped>.”。确保句号出现在末尾且日期写成“month day, year”不带扩展空格的形式。

补充题答案

6.15

```
SELECT DISTINCT date_ordered "ORIGINAL",
     date_ordered + 21 "PROJECTED"
FROM s_ord
WHERE date_ordered = TO_DATE('31 AUG-1992', 'DD-MON-YYYY');
ORIGINAL  PROJECTED
--------- ---------
31-AUG-92 21-SEP-92
```

6.16

```
SELECT DISTINCT date_ordered "ORIGINAL",
     ADD_MONTHS(date_ordered, 2) "PROJECTED"
FROM s_ord
WHERE date_ordered = TO_DATE('31-AUG-1992', 'DD-MON-YYYY');
ORIGINAL  PROJECTED
--------- ---------
31-AUG-92 31-OCT-92
```

6.17

```
SELECT date_ordered, date_shipped,
     date_shipped - date_ordered "BETWEEN"
FROM s_ord
WHERE TO_CHAR(date_ordered, 'MON') = 'AUG'
ORDER BY date_ordered;
DATE_ORDE DATE_SHIP    BETWEEN
--------- --------- ---------
28-AUG-92 17-SEP-92        20
31-AUG-92 10-SEP-92        10
31-AUG-92 10-SEP-92        10
31-AUG-92 10-SEP-92        10
31-AUG-92 18-SEP-92        18
31-AUG-92 15-SEP-92        15
```

6.18

```
SELECT TO_CHAR(SYSDATE, 'fmDAY fmMONTH DD, YYYY') "DAY",
     TO_CHAR(SYSDATE, 'HH12:MI:SS AM') "12 HOUR TIME",
     TO_CHAR(SYSDATE, 'HH24:MI:SS AM') "24 HOUR TIME"
FROM DUAL;
DAY                            12 HOUR TIME  24 HOUR TIME
------------------------------ ------------ ----------------
SUNDAY JANUARY 30, 2000        1:04:37 PM    13:04:37 PM
```

6.19

```
SELECT DISTINCT TO_CHAR(date_ordered, 'fmDDTH') || ' of ' ||
     TO_CHAR(date_ordered,'fmMMTH')  || ' MONTH' "NUMBERS",
     TO_CHAR(date_ordered, 'fmDDSPTH ) || ' of ' ||
     TO_CHAR(date_ordered, 'fmMONTH') "WORDS"
FROM s_ord
WHERE TO_CHAR(date_ordered, 'MON') = 'SEP';
NUMBERS                    WORDS
-------------------------- --------------------------
1ST of 9TH MONTH           FIRST of SEPTEMBER
2ND of 9TH MONTH           SECOND of SEPTEMBER
3RD of 9TH MONTH           THIRD of SEPTEMBER
4TH of 9TH MONTH           FOURTH of SEPTEMBER
7TH of 9TH MONTH           SEVENTH of SEPTEMBER
8TH of 9TH MONTH           EIGHTH of SEPTEMBER
9TH of 9TH MONTH           NINTH of SEPTEMBER
```

6.20

```
SELECT DISTINCT TO_CHAR(date_ordered, 'MM/YY') "FIRST WAY",
     TO_CHAR(date_ordered, 'FMDD MONTH, YYYY') "SECOND WAY"
FROM s_ord
WHERE TO_CHAR(date_ordered, 'MON') = 'AUG';
FIRST WAY          SECOND WAY
------------------ ----------------------------
08/92              28 AUGUST, 1992
08/92              31 AUGUST, 1992
```

6.21

```
SELECT LAST_DAY(SYSDATE) "CURRENT",
     LAST_DAY(ADD_MONTHS(SYSDATE, -2)) "2PREVIOUS",
     LAST_DAY(ADD_MONTHS(SYSDATE, 1)) "NEXT"
FROM DUAL;
CURRENT   2PREVIOUS NEXT
--------- --------- ---------
31-JAN-00 30-NOV-99 29-FEB-00
```

6.22

```
SELECT DISTINCT date_shipped,
     NEXT_DAY(date_shipped, 'FRIDAY') "NEXT FRIDAY"
FROM s_ord
WHERE date_shipped > TO_DATE('20-SEP-1992','DD-MON-YYYY');
DATE_SHIP NEXT FRID
--------- ---------
21-SEP-92 25-SEP-92
22-SEP-92 25-SEP-92
23-SEP-92 25-SEP-92
28-SEP-92 02-OCT-92
```

6.23

```
SELECT ROUND(MONTHS_BETWEEN(TO_DATE('25-DEC-02'), SYSDATE))
     "UNTIL CHRISTMAS 2002"
FROM DUAL;
UNTIL CHRISTMAS 2002
--------------------
                  35
```

6.24

```
SELECT TO_CHAR(SYSDATE,'HH24:MI PM') "GREENWICH",
  TO_CHAR(NEW_TIME(SYSDATE,'GMT','CST'), 'HH24:MI AM') 'CHICAGO",
  TO_CHAR(NEW_TIME(SYSDATE,'GMT','MST'), 'HH24:MI AM') "DENVER"
FROM DUAL;
GREENWICH  CHICAGO       DENVER
---------- ------------- ------------------------- -
15:33 PM   09:33 AM      08:33 AM
```

6.25

```
SELECT DISTINCT date_shipped,
     LEAST(date_shipped, TO_DATE('15-SEP-1992','DD-MON-YYYY'),
     TO_DATE('20-SEP-1992', 'DD-MON-YYYY')) "FIRST",
     GREATEST(date_shipped,
     TO_DATE('15-SEP-1992','DD-MON-YYYY'),
     TO_DATE('20-SEP-1992', 'DD-MON-YYYY')) "LAST"
FROM s_ord
WHERE TO_CHAR(date_shipped, 'MON') = 'SEP';
DATE_SHIP FIRST     LAST
--------- --------- ---------
08-SEP-92 08-SEP-92 20-SEP-92
10-SEP-92 10-SEP-92 20-SEP-92
15-SEP-92 15-SEP-92 20-SEP-92
17-SEP-92 15-SEP-92 20-SEP-92
18-SEP-92 15-SEP-92 20-SEP-92
21-SEP-92 15-SEP-92 21-SEP-92
22-SEP-92 15-SEP-92 22-SEP-92
23-SEP-92 15-SEP-92 23-SEP-92
28-SEP-92 15-SEP-92 28-SEP-92
```

6.26

```
SELECT 'The order from customer ' || customer_id || ' ' ||
     TO_CHAR(date_ordered, 'fmMONTH DD, YYYY') ||
       was shipped on ' ||
     TO_CHAR(date_shipped, 'fmMONTH DD, YYYY') || '.' "ORDERS'
FROM s_ord
WHERE TO_CHAR(date_ordered, 'MON') = 'AUG'
ORDER BY customer_id;
```

```
ORDERS
-------------------------------------- - -- --------- --- ------------ ----
The order from customer 201 AUGUST 28, 1992 was shipped on SEPTEMBER 17, 1992.
The order from customer 202 AUGUST 31, 1992 was shipped on SEPTEMBER 10, 1992.
The order from customer 203 AUGUST 31, 1992 was shipped on SEPTEMBER 18, 1992.
The order from customer 204 AUGUST 31, 1992 was shipped on SEPTEMBER 10, 1992.
The order from customer 205 AUGUST 31, 1992 was shipped on SEPTEMBER 15, 1992.
The order from customer 210 AUGUST 31, 1992 was shipped on SEPTEMBER 10, 1992.
```

第 7 章　复合查询和集合运算符

到目前为止考虑的所有查询都是使用单一的 SELECT 语句从一个或多个表中检索数据。本章将讨论嵌套查询，它允许我们根据另一个查询的结果检索数据。还有，我们将学习嵌套查询的应用以在其他表的数据基础上操纵表。最后，学习集合运算符和它们的某些应用。

7.1　子查询

嵌套查询或子查询允许我们从表中检索数据，这些数据行满足 WHERE 条件，该条件取决于另一个 SELECT 语句返回的值。嵌套查询的通常语法是在一个 SELECT 语句内，包含着另一个 SELECT 语句（见图 7-1）。第一个 SELECT 语句有时称为主查询或外层查询。同样，嵌套的查询有时称为子查询或内层查询。

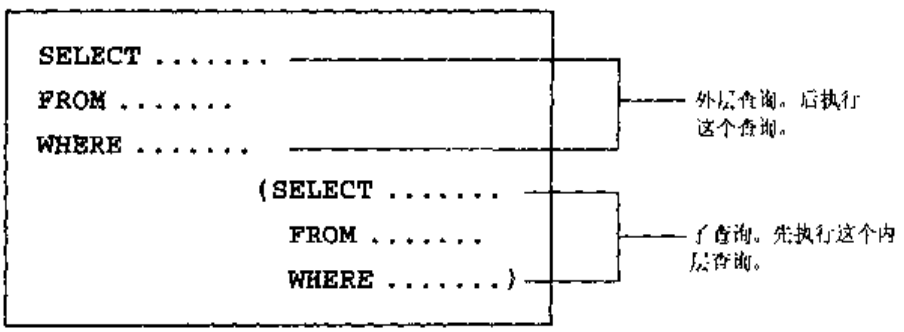


图 7-1　嵌套查询或子查询的基本语法

正如图 7-1 所示，子查询必须在圆括号内，而且必须出现在外层查询的 WHERE 子句条件的右手边。从上面的语法图中，我们仅看到了一个子查询；然而，尽管不太可能发生，但子查询嵌套可以高达 16 层。不管子查询的嵌套数是多少，执行的顺序总是从最内层的查询到最外层的查询。有时，子查询按它们返回的行数（一对多行）或比较的列数（单列或多列）来分类。

7.1.1　单一行的子查询

从一个表中返回单一值的子查询称为单一行子查询。在最外层查询的 WHERE 子句的条件中使用的运算符称为单一行比较运算符（见表 7-1）。

表 7-1　单一行比较运算符

运算符	意义
=	相等
< >	不等或者不相同
>	大于
>=	大于等于
<	小于
<=	小于等于

例 7.1

使用 S_EMP 表，显示所有工资大于平均工资的雇员的姓氏、名字和工资。

为了回答这个查询，首先，我们需要确定所有雇员的平均工资；其次，需要找到哪些雇员的工资大于平均工资。

用如下的查询，确定所有雇员的平均工资：

```
SELECT AVG(SALARY)
FROM s_emp;

AVG(SALARY)
-----------
1542.48
```

利用这个查询结果，再发出如下命令，可以查到哪些雇员的工资大于平均工资：

```
SELECT first_name, last_name, salary
FROM s_emp
WHERE salary > 1542.48;
FIRST_NAME               LAST_NAME                 SALARY
------------------------ ------------------------- ------
Carmen                   Martin                      4500
Doris                    Smith                       2450
Michael                  Norton                      2400
Mark                     Quentin                     2450
Joseph                   Roper                       2550
Molly                    Brown                       1600
Roberta                  Hawkins                     1650
Antoinette               Catskill                    1700

8 rows selected.
```

可将这两个 SELECT 语句组合成如下子查询的单一查询：

```
SELECT first_name, last_name, salary  <-- 外层查询
FROM s_emp
WHERE salary > (SELECT AVG(SALARY)  <-- 子查询或者内层查询
               FROM s_emp);
```

注意，子查询返回一个单一值，该平均值用于计算外层查询 WHERE 条件表达式的值时使用。

7.1.2 多行子查询

多行子查询返回一行或者多行数据，可以使用表 7-2 中的一个多行比较运算符将返回的行加入到外层查询的 WHERE 子句中。

表 7-2 多行比较运算符

运算符	意义
IN	等于子查询返回值中的任一值，则为 TRUE
NOT IN	不等于或不同于子查询返回值中的任一值，则为 TRUE
ANY	一个值与子查询返回值中的每一个值比较。只要有一个成立，则为 TRUE
ALL	一个值与子查询返回值中的每一个值比较，都成立时，则为 TRUE

例 7.2

在 s_dept 表中，显示至少有一位雇员的部门的名称。

因为内层查询可能检索到一行以上的查询结果，因此在外层查询的 WHERE 子句中，需要使用 IN 运算符。注意，外层查询的 SELECT 子句中 DISTINCT 的使用是为了防止显示中有重复行出现。

```
SELECT DISTINCT name
FROM s_dept
WHERE id IN (SELECT dept_id
             FROM s_emp);
```

这个运算符必须是一个多行比较运算符

多行子查询

```
NAME
-------------------------
Administration
Finance
Operations
Sales
```

7.1.3　多列子查询

多列子查询允许内层查询和外层查询之间多列数据的比较。外层查询的 WHERE 条件列出了用于比较的列名，列名之间用逗号分开。子查询从内层表中选择对应的列。

这种类型子查询的语法如图 7-2 所示

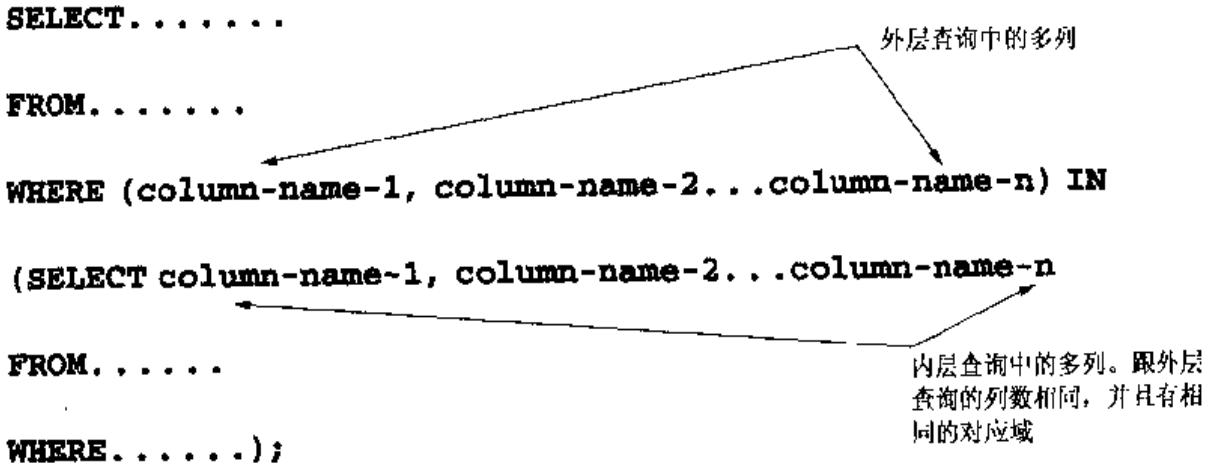


图 7-2　多列子查询的语法

例 7.3

根据表 S_EMP 显示经理 ID（manager-id）为 10 且工资和职务相同的所有雇员的名字、姓氏、部门号、工资和开始工作的日期

回答这个查询之前，将如下元组插入到 S_EMP 表中（参见附录 D 脚本中 S_EMP 表的结构）。

```
INSERT INTO s_emp VALUES ('300', 'Smith', 'Albert', 'smithal',
    TO_DATE('09-MAY-1998','DD-MON-YYYY'), NULL, '10', 'Sales
Representative',
    '31', 1100, 10);
INSERT INTO s_emp VALUES ('301', 'Jones', 'Jenny', 'jonesjen',
    TO_DATE('31-AUG-1999', DD-MON YYYY'), NULL, '10', 'Sales
Representative',
    '31', 1100, 10);
INSERT INTO s_emp VALUES ('302', 'Barker', 'Joel', 'barkerjo',
    TO_DATE('09-JUL-1999','DD-MON YYYY'), NULL, '10', 'Sales
Representative',
    '31', 1400, NULL);
INSERT INTO s_emp VALUES ('303', Armstrong', 'Marta', 'armstmar',
    TO_DATE('26-MAR-1999','DD-MON-YYYY'), NULL, '10', 'Sales
Representative',
    31', 1100, 10);
INSERT INTO s_emp VALUES ('304', 'Nichols', 'Jim', nichojim',
    TO_DATE('11-DEC 1999','DD-MON-YYYY'), NULL, '10', 'Sales
```

```
Representative',
    '31', 1400, 10);
INSERT INTO s_emp VALUES ('305 , 'Ritchie', 'Diane', ritchdi',
    TO_DATE('13-FEB-1998','DD-MON-YYYY'), NULL,  10',  'Sales
Representative',
    '31', 1100, NULL);
INSERT INTO s_emp VALUES ('306',  Good', 'Gwen', 'goodgw',
    TO_DATE('01-SEP-1998','DD-MON-YYYY'), NULL,  10',  'Sales
Representative',
     31 , 1400, NULL);
COMMIT;
```

该查询说明了多列子查询的使用。在这种情况下，首先找所有 manager_id = 10 的雇员，然后，从内查询返回的雇员子集中，检索到有相同职务和工资的所有雇员。

```
SELECT first_name, last_name, dept_id, salary, start_date
FROM s_emp
WHERE (salary, title) IN
             (SELECT salary, title
              FROM s_emp
              WHERE manager_id = '10 );
```

FIRST_NAME	LAST_NAME	DEP	SALARY	START_DAT
Blaine	Stephenson	45	860	17-MAR-91
Bonnie	Dancer	45	860	17-MAR-91
Albert	Smith	31	1100	09-MAY-98
Marta	Armstrong	31	1100	26-MAR-99
Diane	Ritchie	31	1100	13-FEB-98
Jenny	Jones	31	1100	31-AUG-99
Colin	Henderson	31	1400	14-MAY-90
Joel	Barker	31	1400	09-JUL-99
Gwen	Good	31	1400	01-SEP-98
Jim	Nichols	31	1400	11-DEC-99

7.2 相关查询

相关查询是一种嵌套查询，与前两节中谈到的子查询相比，它提供了一种更有效地检索数据的机制。与子查询不同，在相关查询中，内层查询总是引用外层查询 FROM 子句中提及的表。另外，相关查询与子查询不仅仅是执行顺序不同，而且查询执行的次数也不相同。相关查询利用表的别名来引用外层查询中指定的值。在相关查询的内容中，一些作者要用表的别名标识列，这样的表别名称为相关变量或相关名。本书中这些术语可以互换使用。相关查询的基本语法如图 7-3 所示。

在所有相关查询中，执行总是从外层查询开始。外层查询选择外层表的各个行，并把它们作为候选行。对候选行中的每一行，相关内层查询执行一次。在内层查询执行期间，系统

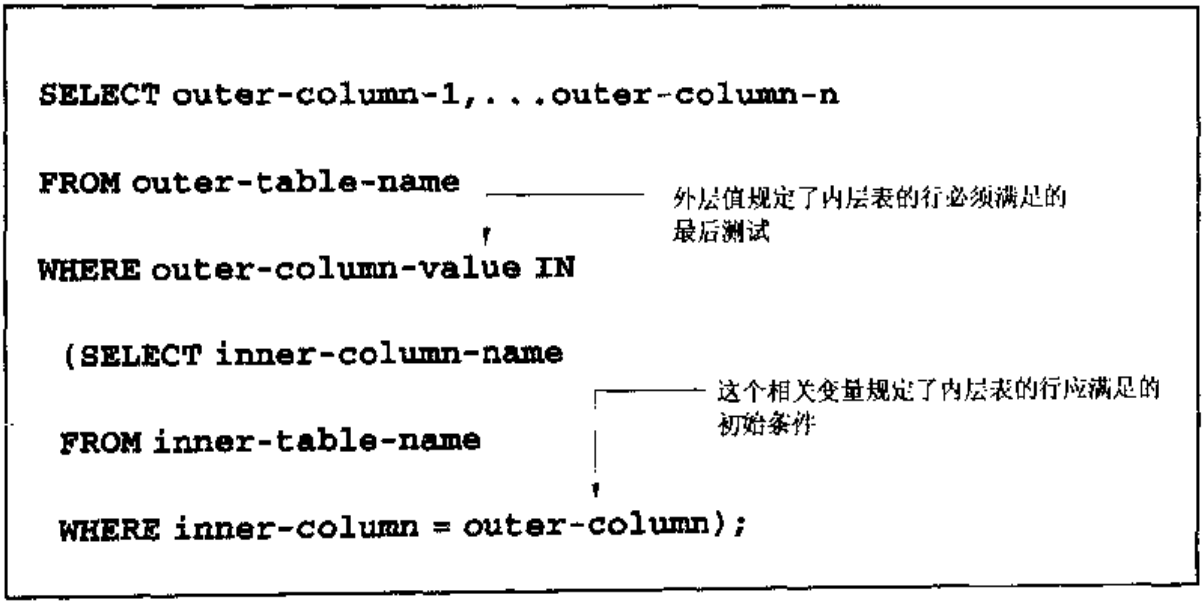


图 7-3　相关查询的语法

根据外层指定的列值寻找内层满足 WHERE 条件的行。内层表中满足该条件的所有行形成一个临时集。然后，系统再对存储在临时集中的行测试外层条件。所有满足外层条件的行被显示出来。这个过程一直继续到所有候选行处理完为止。

例 7.4

显示所有在 1992 年 8 月 31 日发出定单的顾客。

下面是能让我们检索数据和结果的相关查询：

```
SELECT name
FROM s_customer outer
WHERE TO_DATE('31-AUG-1992','DD-MM-YYYY')
  IN ( SELECT date_ordered
       FROM s_ord inner
       WHERE inner.customer_id = outer.id);
```

相关变量

```
NAME
--------------------
Deportivo Caracas
New Delhi Sports
Ladysport
Kim's Sporting Goods
Futbol Sonora
5 rows selected.
```

在这个查询中，我们使用了表别名 outer 和 inner，分别表示外层查询和内层查询的表。该查询的求解过程如下：

- 外层查询每次从外层表中选择一行，这些行中的每一行都成为候选行。
- 候选行的值存储在外层表的别名中。
- 对这个特殊行执行子查询。系统通过整个内层表寻找有相同 outer.id 值的行，在这种情况下，就是当前行的 customer ID。所有有相同 customer ID 的行形成一个临时集。
- 使用上一步的临时集，系统用外层查询的条件，去找定单日期为“31-AUG-1992”的所有行。满足该条件的行显示在结果表中。
- 系统重复步骤 1 到步骤 4，直到外层表的所有行被测试完为止。

7.3 使用子查询创建表

有时，我们希望拷贝表的结构但不要它的数据，或者希望只拷贝表的一部分并带有相应的数据。可以使用 CREATE TABLE 命令的各种变体实现这一目的。图 7-4 说明了使用子查询的 CREATE TABLE 语句的变体的基本语法。

```
CREATE TABLE table-new-name AS
SELECT column-name-1, column-name-2,...column-name-n
FROM table-name
WHERE column-name = (SELECT column-name
                     FROM table-name
                     WHERE condition);
```

图 7-4 使用带子查询的 CREATE TABLE 命令来拷贝一个表或表的一部分

7.3.1 拷贝表的结构

为了拷贝一个表的结构而不要它的数据，可以使用 * 号和一个永远也不能满足的布尔条件表达式。下面的例子说明了这一点。

例 7.5

拷贝 S_EMP 表的结构，而不要它的数据。将新表称为 S_WORKER。

对应的 CREATE TABLE 命令如下：

```
CREATE TABLE s_worker ←—— 正创建的新表名称

AS SELECT * ←—————————— 这个*号允许我们选择 S_EMP 表的所有列

FROM s_emp

WHERE 1 <> 1; ←—————————— 永远不能满足这个布尔条件表达式
```

这个 * 号，允许我们选择 S_EMP 表的所有列。注意表达式 1 < > 1 的使用，它是使表的任何行都不能满足的布尔条件。由于没有检索到行，所以仅拷贝了表的结构。

7.3.2 拷贝表中的若干列及其数据

为了拷贝表中的某些列和它们的数据，我们只要指定该列和在 CREATE TABLE 命令中正在拷贝的表中的行需要满足的条件即可。可以用下面的例子说明这一点。

例 7.6

创建一个表，它包含有 VP 职务的雇员的名字、姓氏和部门号 新表名为 Vice_President。

```
CREATE TABLE Vice_President AS
SELECT first_name, last_name, dept_id
FROM s_emp
WHERE id IN (SELECT id
             FROM s_emp
             WHERE title LIKE '%VP%');
```

为了验证拷贝的结构和数据的正确性，我们可以使用下面的命令：

```
DESCRIBE Vice_President
Name                                      Null?    Type
----------------------------------------- -------- ----
FIRST_NAME                                         VARCHAR2(20)
LAST_NAME                                 NOT NULL VARCHAR2(20)
DEPT_ID                                            VARCHAR2(3)

SELECT * FROM vice_president;
FIRST_NAME           LAST_NAME                 DEP
-------------------- ------------------------- ---
Doris                Smith                     41
Michael              Norton                    31
Mark                 Quentin                   10
Joseph               Roper                     50

4 rows selected.
```

7.4 利用子查询更新表

利用子查询来更新表的内容是可能的。这个命令的语法如下：

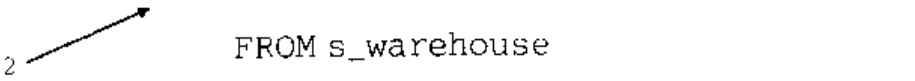
```
UPDATE table-name
SET column = (SELECT column-name
              FROM table-name
              WHERE condition)
WHERE column-name = (SELECT column-name
                     FROM table-name
                     WHERE condition);
```

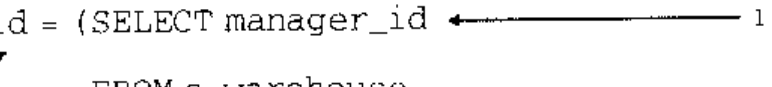
例 7.7

假设斯洛伐克的仓库经理辞职了，由副总裁 Michael Norton 临时代替。更新 S_EMP 表以反应这种变化。

下面是反映这种变化的 UPDATE 命令：

```
UPDATE s_emp
SET manager_id = (SELECT id
                  FROM s_emp
                  WHERE last_name = 'Norton')
WHERE manager_id = (SELECT manager_id
                    FROM s_warehouse
                    WHERE country = 'Slovakia');
```





该命令的处理是按由底向上的方式进行的。观察 1 号箭头所指的子查询，它从 s_warehouse表查到了斯洛伐克经理的标识号（manager_id）系统利用这个 manager_id，从 S_EMP 表中检索其经理是斯洛伐克仓库经理的雇员（见 2 号箭头）。最后，使用 3 号箭头所指的子查询返回的值更新 S_EMP 表中相应的行。

7.5 利用子查询向表内插入数据

子查询也允许我们使用 INSERT INTO 表的各种变体向表内增加行。与子查询一起使用时，将行从一个表插入到另一个表中是可能的。这个命令的语法如下：

```
INSERT INTO table-name (column-name-1,...,column-name-n)
SELECT column-name-1,...,column-name-n
FROM table-name
WHERE column-name = (SELECT column-name
                     FROM table-name
                     WHERE condition);
```

例7.8

在S_EMP表中，将所有在10号部门工作的雇员插入到例7.5的S_WORKER表中。

```
INSERT INTO s_worker
SELECT *
FROM s_emp
WHERE dept_id='10';
```

7.6 利用子查询从表中删除行

使用子查询和DELETE FROM命令，可以使我们根据另一个表的信息从一个表中删除若干行。这个命令的基本语法如下：

```
DELETE FROM table-name
WHERE column-name = (SELECT column-name
                     FROM table
                     WHERE condition);
```

例7.9

假定由于业务不足，位于斯洛伐克的仓库已经关闭。在表S_EMP中，删除所有在这个仓库工作的雇员。在回答这个问题之前，刷新运动用品表。

相应的DELETE命令如下：

```
DELETE FROM s_emp
WHERE manager_id = (SELECT manager_id
                    FROM s_warehouse
                    WHERE country = 'Slovakia');
```

7.7 集合运算符

集合运算符UNION(并)、INTERSECT(交)和MINUS(差)允许将单独的SELECT语句的结果组合起来。它们分别与SQL集合运算符UNION、INTERSECTION和DIFFERENCE等价。在这三个运算符中，UNION(并)操作是最有效和最有趣的。

7.7.1 UNION 运算符

UNION 运算符允许我们把多个 SELECT 语句的查询结果，组成单一的 SELECT 语句。图 7-5 展示了这个命令的基本语法。

```
SELECT table-1-column-name-1,...,table-1-column-name-n
FROM table-1
WHERE condition-1
UNION ←——————————— 关键字必须出现在两个SELECT语句之间
SELECT table-2-column-name-1,...,table-2-column-n
FROM table-2
WHERE condition-2;
```

图 7-5 两个 SELECT 语句并运算的语法

在两个表合并之前，用户需要确信 SELECT 语句的查询结果是可以进行并运算的。也就是说，每个 SELECT 语句的查询结果需要满足以下条件：

- 在每个中间结果表中，列数必须相同。
- 对应列的数据类型必须相同，但不需要列名相同。在形成并运算时，Oracle 使用第一个 SELECT 语句确定结果表的列标题。
- 假如使用 ORDER BY 子句，它必须是最后一个 SELECT 语句的最后子句，而且只能出现一次。
- ORDER BY 子句必须引用它相关的列号，不能用列名。

下面的例子说明了 UNION 运算符的用法。

例 7.10

显示所有 region_id 为 1 的顾客姓名和在该地区的部门名称。

参与 UNION 运算的 SELECT 语句和它们相应的结果显示如下。注意，尽管没有使用 ORDER BY 子句来指定，但结果是排好序的。UNION 运算的结果总是排序的，因为 Oracle 执行内部排序，以辨认重复行，这是 UNION 理论上定义的需要。

```
SELECT name
FROM s_customer
WHERE region_id ='1'
UNION
SELECT name
FROM s_dept
WHERE region_id ='1';
NAME
--------------------
Administration
Beisbol Si!
Finance
Ladysport
Operations
Sales
Sports Emporium
Sports Retail
.
.
17 rows selected.
```

7.7.2　INTERSECT 运算符

INTERSECT 运算符允许找出两个表共有的行。这个交运算符也要放在两个 SELECT 语句之间。交运算符的使用语法如下：

```
SELECT table-1-column-name-1,...,table-1-column-name-n

FROM table-1

WHERE condition-1

INTERSECT ←——————————— 关键词 INTERSECT 必须位于两个 SELECT 语句中间

SELECT table-2-column-name-1,...,table-2-column-n

FROM table-2

WHERE condition-2;
```

SELECT 语句查询的中间结果必须满足并兼容性条件。下面的例子说明了这个运算符的用法。

例 7.11

显示所有被指派了客户的销售代表的 ID。

这个相应的查询语句及其查询结果如下：

```
COLUMN sales_rep_id HEADING 'Sales Reps  FORMAT A11
SELECT sales_rep_id
FROM s_customer
INTERSECT
SELECT id
FROM s_emp;

Sales Reps
-----------
11
12
13
14

4 rows selected.
```

7.7.3 MINUS 运算符

MINUS 运算符允许我们确定存在于一个表中但不存在于另一表中的行。不像 UNION 和 INTERSECT 两个运算符，MINUS 运算符没有交换性。也就是说，表 A MINUS 表 B 的结果通常与表 B MINUS 表 A 的结果不同。下面的例子说明 MINUS 运算符的使用。

例 7.12

由于内部培训，有些销售代表还没有被指派客户，显示这些销售代表的姓名。

这个相应的查询语句及其查询结果如下：

```
SELECT id
FROM s_emp
MINUS
SELECT sales_rep_id
FROM s_customer;

ID
---
1
10
```

```
15
16
.
.
.
20 rows selected.
```

问题与答案

7.1　显示那些 ID 为 2 的雇员所在的部门名。

为获得这个信息，查询语句及其结果如下所示。

```
SELECT name
FROM s_dept
WHERE id = (SELECT dept_id
            FROM s_emp
            WHERE id = '2');
NAME
--------------------
Operations
```

在这个查询中，子查询返回了给出的雇员的部门 ID。然后主查询利用这个部门 ID 显示相应的部门名。

7.2　显示某些雇员的姓氏、名字和工资，这些雇员跟姓为 'Brown' 的员工在同一部门。

这个查询和它的结果如下所示。

```
SELECT first_name, last_name, salary
FROM s_emp
WHERE dept_id IN (SELECT dept_id
                  FROM s_emp
                  WHERE last_name = 'Brown');
FIRST_NAME           LAST_NAME            SALARY
-------------------- -------------------- ---------
Doris                Smith                   2450
Molly                Brown                   1600
George               Brown                    940
Elaine               Hardwick                1400
```

在这种情况下，子查询返回的结果多于一行，因为在同一部门，姓为 'Brown' 的不止有一个雇员。可以看到有两名雇员都姓 'Brown'，他们在同一部门工作。因此，在主查询中需要用 IN 运算符来代替等号。

7.3　显示销售代表 Andre Dameron 的所有客户的名称、所在城市和国家。

该查询语句及其结果如下：

```
SELECT name, city, country
  FROM s_customer
```

```
WHERE sales_rep_id = (SELECT id
                      FROM s_emp
                      WHERE last_name = 'Dameron');

NAME                 CITY                 COUNTRY
-------------------- -------------------- --------------
Toms Sporting Goods  Harrisonburg         US
Athletic Attire      Harrisonburg         US
.
.
5 rows selected.
```

7.4 显示任职于一个现有部门的雇员的姓名。

这个查询及其运行结果如下：

```
SELECT first_name, last_name
FROM s_emp
WHERE dept_id = ANY (SELECT id FROM s_dept);
FIRST_NAME           LAST_NAME
-------------------- --------------------
Carmen               Martin
Doris                Smith
Michael              Norton
Mark                 Quentin
Joseph               Roper
.
.
24 rows selected.
```

在这个例子中，使用了一个关键字 ANY，它可以与比较运算符中的任一个运算符一起使用，无论该子查询返回一行还是多行均是如此。当 ANY 与比较运算符一起使用时，若条件表达式对子查询返回的某一个值是真时，则布尔条件表达式值为真。在这种情况下，子查询简化如下：

```
SELECT first_name, last_name
FROM s_emp
WHERE dept_id  ANY ('Finance', 'Sales', 'Operations',
                    'Administration');
```

7.5 找出工资最低的雇员。

```
SELECT first_name, last_name
FROM s_emp
WHERE salary <= ALL (SELECT salary
                     FROM s_emp);
FIRST_NAME           LAST_NAME
-------------------- ----------
Donald               Patterson
Roger                Pearl
```

在这个习题中有关键字 ALL，不管子查询返回一行还是多行数据，它均能够跟任意一

个比较运算符一起使用。当使用 ALL 时，若表达式对子查询返回的所有值为真，则条件的值为真。在这种情况下，有两个雇员的工资低于或等于所有其他雇员的工资。

7.6　显示工资和佣金百分率与雇员 Jason Sanders 相同的雇员的姓名和部门 ID。在回答这个问题之前，在表 S_EMP 中插入如下元组：

```
INSERT INTO s_emp (id, last_name, first_name, userid, start_date,
manager_id, title, dept_id, salary, commission_pct)
VALUES ('514', 'Chloe', 'Wendt', 'wendtch',TO_DATE( 18-FEB-1991','DD-
MON-YYYY'),'3', 'Sales Representative','33', 1515, 10);
INSERT INTO s_emp (id, last_name, first_name, userid, start_date,
manager_id, title, dept_id, salary, commission_pct) VALUES ('611',
'Ritchie', 'Erin', 'ritchri',TO_DATE('18-FEB-1991','DD-MON-YYYY'),'3',
 Sales Representative','33', 1515, 10);
```

这个查询及其运行结果如下：

```
SELECT last_name,first_name, dept_id
FROM s_emp
WHERE (salary, commission_pct)
IN (SELECT salary, commission_pct
    FROM s_emp
    WHERE dept_id IN (SELECT dept_id FROM s_emp
                      WHERE last_name = 'Sanders'));
```

```
LAST_NAME                 FIRST_NAME                DEP
------------------------- ------------------------- ---
Sanders                   Jason                     33
Ritchie                   Erin                      33
Chloe                     Wendt                     33
```

注意，在这个查询中，最内层的子查询检索到了 Jason Sanders 的部门 ID，上一层子查询检索到了这个雇员的工资和佣金百分率。最后，外层查询返回了所有与 Jason Sanders 的工资和佣金百分率匹配的雇员信息。

7.7　显示在 1992 年 8 月 31 日发出定单的所有顾客的姓名、所在城市、国家和销售代理。这个查询和它的结果如下所示。为解答这个问题，我们需利用这样一个事实，即每张定单上有客户和他的销售代表的名称或姓名。注意，这结果被格式化了，以适应页的宽度。

```
SELECT name, city, country, sales_rep_id
FROM s_customer
WHERE (id, sales_rep_id) IN
(SELECT customer_id, sales_rep_id AS 'Rep"
 FROM s_ord
 WHERE date_ordered='31-AUG-1992');
```

```
NAME                      CITY                      COUNTRY                   Rep
Ladysport                 Seattle                   US                        11
Futbol Sonora             Nogales                   Mexico                    12
```

7.8　对于每位销售代表，若他的定单合计大于合计平均值，则显示销售代表号和他或她的定单合计金额。

这个查询和查询结果如下所示。

```
SELECT sales_rep_id AS "Rep", SUM(total) AS "Total Orders"
FROM s_ord
GROUP BY sales_rep_id
HAVING SUM(total) > (SELECT AVG(total)
                     FROM s_ord);
```

```
Rep Total Orders
--- ------------
11     1629443.4
13        182000
14      158529.6
```

在这种情况下，子查询计算了所有定单合计的平均值。然后，这个结果被用在主查询中，以显示其定单合计超过平均值的每个销售代表。

7.9 显示定单超过定单平均值的所有顾客的姓名、所在城市、国家和信誉等级。

这个查询和查询结果如下所示。注意，为检索到所需要的顾客信息，使用了子查询的返回值并结合了一个连接条件。为了适应需要修改了输出。

```
SELECT name, city, country, credit_rating AS "Rating"
FROM S_customer, s_ord
WHERE total > (SELECT AVG(total)
               FROM s_ord)
               AND s_ord.customer_id = s_customer.id;
```

```
NAME                 CITY                 COUNTRY              Rating
-------------------- -------------------- -------------------- ---------
Ladysport            Seattle              US                   EXCELLENT
Helmut's Sports      Prague               Czechoslovakia       EXCELLENT
Hamada Sport         Alexandria           Egypt                EXCELLENT
Sports Emporium      San Francisco        US                   EXCELLENT
```

7.10 使用相关查询，显示至少有一个雇员的所有部门名。

这个查询及结果如下所示。注意，在这个查询中使用了 EXISTS 运算符。当我们感兴趣的是了解而非某一行的值时，不管子查询有无返回行，都可以使用 EXISTS 运算符。若子查询返回任意行，EXISTS 运算为 TRUE。DISTINCT 子句的使用避免了在结果中出现重复的部门名。

```
SELECT DISTINCT name
FROM s_dept D
WHERE EXISTS (SELECT E.dept_id
              FROM s_emp E
              WHERE E.dept_id = D.id);
```

```
NAME
-------------
Administration
Finance
Operations
Sales
```

7.11　显示没有客户的所有雇员的姓氏和名字。

这个查询及结果如下所示。

```
SELECT last_name, first_name
FROM s_emp Outer
WHERE id NOT IN (SELECT sales_rep_id
                 FROM s_customer Inner
                 WHERE inner.sales_rep_id = outer.id);
LAST_NAME              TITLE
---------------------- ---------------------
Martin                 President
Smith                  VP, Operations
Norton                 VP, Sales
Quentin                VP, Finance
.
.
22 rows selected.
```

7.12　存在没有销售代表的国家吗？假若有的话，显示国家的名称。使用相关查询和 NOT EXISTS 运算符。

这个查询及结果如下所示。

```
SELECT country
FROM s_customer C
WHERE NOT EXISTS (SELECT id
                  FROM s_emp E
                  WHERE E.id = C.sales_rep_id);

COUNTRY
--------------------
Nigeria
```

7.13　写一个查询，显示位于或者指派到委内瑞拉的顾客和销售代表。使用 UNION 运算符。

这个查询和它的结果如下所示。

```
SELECT C.name
FROM s_customer C
WHERE country = 'Venezuela'
UNION
SELECT last_name
FROM s_emp
WHERE id IN (SELECT sales_rep_id
FROM s_customer
WHERE country = 'Venezuela');
NAME
--------------------
Deportivo Caracas
Gilson
```

7.14 显示委内瑞拉和俄罗斯的销售代表的 ID。用 UNION 运算符。

```
SELECT id
FROM s_emp E
WHERE id IN (SELECT sales_rep_id
             FROM s_customer C
             WHERE country = 'Venezuela')
UNION
SELECT id
FROM s_emp E
WHERE id IN (SELECT sales_rep_id
             FROM s_customer C
             WHERE country = 'Russia');
ID
---
11
12
```

7.15 创建一个表，它包括信誉等级为‘POOR’的所有客户的姓名、所在城市、州、国家和地区。表名为 Low_Rating。

相应的 CREATE TABLE 命令如下所示。

```
CREATE TABLE low_rating
AS SELECT name, city, country, region_id
FROM s_customer
WHERE credit_rating = 'POOR';
```

7.16 创建一个名为 PC_users 的表，它有如下的列：姓氏、名字和 SG 公司所有雇员的用户 ID。要确保没有数据拷贝到创建的这个表中，并且表已被正确地创建了。

```
CREATE TABLE pc_users AS
SELECT first_name, last_name, userid
FROM s_emp
WHERE id <>id;

DESC pc_users;
Name                                      Null?    Type
----------------------------------------- -------- ----
FIRST_NAME                                         VARCHAR2(20)
LAST_NAME                                 NOT NULL VARCHAR2(20)
USERID                                    NOT NULL VARCHAR2(8)
```

7.17 创建一个与 s_dept 结构相同的表，但是按下面表格中指示的那样，重新命名列名。新表名为 WORK_UNIT。

S_DEPT 表中的名称	Work_Unit 表中的名称
Id	Unit_id
Name	Unit_name
Region_id	Global_area

创建表 WORK _ UNIT 的语句如下：

```
CREATE TABLE work_unit (unit_id, unit_name, global_area)
AS SELECT id, name, region_id
FROM s_dept;
```

为验证创建的表是否有所需要的结构，发出如下命令：

```
DESC work_unit;
Name                                    Null?     Type
--------------------------------------- --------- ----
UNIT_ID                                 NOT NULL  VARCHAR2(3)
UNIT_NAME                               NOT NULL  VARCHAR2(20)
GLOBAL_AREA                                       VARCHAR2(3)
```

7.18　前面已经创建了 WORK _ UNIT 表，并且数据也从 s _ dept 表拷贝到此表中。写一个子查询，以从 WORK _ UNIT 表中删除所有的行。

这个相应的子查询如下所示。注意，这个子查询返回 s _ dept 表中的每一个 ID。这些 ID 中的每一个都被用做在表 WORK _ UNIT 中删除的依据。

```
DELETE FROM work_unit
WHERE unit_id IN (SELECT id
                  FROM s_dept);
```

7.19　由于上一年出色地完成了任务，因此，对其经理位于西雅图、华盛顿的所有雇员，每人工资长 1000 美元，写一个相应的查询，以更新这些雇员的信息

相应的查询如下所示。

```
UPDATE s_emp
SET salary = salary + 1000
WHERE manager_id = (SELECT manager_id
                    FROM s_warehouse
                    WHERE city = 'Seattle' AND state = 'WA');
```

补　充　题

7.20　显示姓为 Dameron 的雇员的部门名称。

7.21　显示工资最低的雇员的姓氏、名字和开始工作的日期。不要使用与在问题与答案 7.5 中相同的方法。

7.22　显示为 Michael Norton 工作的所有雇员的名字和姓氏。

7.23　显示所有没有雇员的部门的名称。

7.24　显示所有其销售代表也是仓库经理的客户的姓名、国家和地区。有多少客户满足这个条件？

7.25　显示在 1992 年 7 月进行订购的顾客的名称、所在城市、国家和他们的销售代表。

7.26 显示所有定单总额超过平均定单总额的销售代表的姓名。

7.27 写一个相关查询，显示目前还没有雇员的部门的名称。要求使用 NOT EXISTS 运算符，参考问题与答案中的 7.10 题。

7.28 使用相关查询，显示有客户的所有销售代表的姓氏和名字。

7.29 显示所有其客户的信誉等级为“GOOD”的销售代表的姓名。利用相关查询和 EXISTS 运算符。

7.30 显示公司的所有顾客和销售代表的名称。

7.31 使用 UNION 运算符显示委内瑞拉和俄罗斯的销售代表的姓名。

7.32 删除在哈里森堡、弗吉尼亚有顾客的所有销售代表。在解答这个问题之前，先运行脚本 SG_NO_CONSTRAINTS.SQL。

7.33 删除当前没有任何定单的客户。

补充题答案

7.20

```
SELECT name
FROM s_dept
WHERE id = (SELECT dept_id
            FROM s_emp
            WHERE LAST_NAME = 'Dameron');
```

7.21

```
SELECT last_name, first_name, start_date
FROM s_emp
WHERE salary = (SELECT MIN(salary)
                FROM s_emp);
```

7.22

```
SELECT first_name, last_name
FROM s_emp
WHERE manager_id = (SELECT id
                    FROM s_emp
                    WHERE last_name = 'Norton');
```

7.23

```
SELECT name
FROM s_dept
WHERE id NOT IN (SELECT dept_id
                 FROM s_emp);
```

7.24

```
SELECT name,city,country
FROM s_customer
WHERE (sales_rep_id, region_id) IN
(SELECT manager_id, region_id
 FROM s_warehouse);
There are no sales representatives that are also managers.
```

7.25

```
SELECT name, city, country, sales_rep_id
FROM s_customer
WHERE (id, sales_rep_id) IN
(SELECT customer_id, sales_rep_id
 FROM s_ord
 WHERE date_ordered BETWEEN '01-JUL-1992' AND
                            '31-AUG-1992');
```

7.26

```
SELECT last_name
FROM s_emp
WHERE id IN (SELECT sales_rep_id
             FROM s_ord
             GROUP BY sales_rep_id
             HAVING SUM(total) > (SELECT AVG(total)
                                  FROM s_ord));
```

7.27

```
SELECT name
FROM s_dept D
WHERE NOT EXISTS (SELECT E.dept_id
                  FROM s_emp E
                  WHERE E.dept_id = D.id);
```

7.28

```
SELECT last_name, first_name
FROM s_emp Outer
WHERE id IN (SELECT sales_rep_id
             FROM s_customer Inner
             WHERE inner.sales_rep_id = outer.id)
AND Title = 'Sales Representative';
```

7.29

```
SELECT last_name
FROM s_emp E
WHERE EXISTS (SELECT name
              FROM s_customer C
              WHERE C.sales_rep_id = E.id
              AND C.credit_rating='GOOD');
```

7.30

```
SELECT C.name
FROM s_customer C
UNION
SELECT last_name
FROM s_emp
WHERE title ='Sales Representative';
```

7.31

```
SELECT last_name
FROM s_emp
WHERE id IN (SELECT id
             FROM s_emp E
             WHERE id IN (SELECT sales_rep_id
                          FROM s_customer C
                          WHERE country = 'Venezuela')
UNION
SELECT id
FROM s_emp E
WHERE id IN (SELECT sales_rep_id
             FROM s_customer C
             WHERE country = 'Russia'));
```

7.32

```
DELETE
FROM s_emp
WHERE id IN (SELECT sales_rep_id
             FROM s_customer
             WHERE city = 'Harrisonburg' AND state ='VA');
```

注意：这个查询将产生一个错误，这是因为破坏了数据的参照完整性约束。它只能在SG_NO_Constraints.sql数据库中工作。

7.33

```
DELETE FROM s_customer
WHERE NOT EXISTS (SELECT *
                  FROM s_ord
                  WHERE customer_id = s_customer.id);
```

第 8 章　使用 SQL 的基本安全性问题

本章在 SQL 语言的框架和能力范围内讨论有关数据安全性的一些基本问题，以认证用户和保护数据，防止数据的非法使用。

8.1　数据安全性

在任何公司中，数据都是最宝贵的资源。因此，需要控制、管理和保护数据，并使之处于安全状态。在 RDBMS 的上下文中称为安全性。我们将谈到数据库的保护，以防止未经授权的访问、故意或非故意的泄露、变更或者数据库破坏。尽管在 DBMS 的安全性方面还需要关注其他问题，如物理保护和网络保护，但在本书中，我们仅讨论基于计算机的某些基本对抗手段，特别是将讨论认证和授权问题

8.1.1　认证

认证是引用某种机制，确认用户是不是他或她所声称的人。认证能在操作系统或 RDBMS 级执行。在任何一种情况下，系统管理员（SYSAD）或数据库管理员（DBA）为每个用户建立一个账户或用户名。另外，还为这些账户的用户分配口令。口令是一个由字符、数字或者二者的组合组成的序列。按理说只有系统和它的合法用户知道该口令。RDBMS 以加密的方式将用户名和口令存储在数据字典中。为了访问数据库，用户必须运行一个应用程序，并且用自己的账户或用户名和相应的口令与数据库连接。例如，对于本书的练习，我们分别使用“scott”和“tiger”作为用户名和口令通过 SQL * Plus 应用程序与 Oracle 数据库连接。

由于口令是防止外部人员未经授权使用数据库的第一道防线，所以它的合法拥有者要保守机密。这里建议用户要经常改变自己的口令。口令最少 6 个字符长，由字母、数字和键盘上允许的其他符号组成。用户应避免用普通名称，如绰号或固有名称等。为了在 Oracle 中改变口令，用户可以发出如下命令：

```
ALTER USER user-name IDENTIFIED BY new-password;
```

例 8.1

以 scott/tiger 登录进 P08，并将口令改变为 RJ89。

相应的指令如下：

```
ALTER USER Scott IDENTIFIED BY RJ89;
```

显然，改变了口令以后，用户将不能再使用以前的口令。只要自己愿意，用户可以多次改变自己的口令。

8.1.2 授权

授权可以理解为授予用户权力或特权，允许他们访问该系统或系统内的对象。与系统用户相关的特权的基本类型有系统特权和对象特权。系统特权能使用户访问数据库，对象特权允许用户操纵数据库内的对象。当一个数据库的用户建立后，该用户就跟数据库模式或在数据库中的一组对象联系起来，可以访问它们。系统管理员通过用户的安全域确定和控制用户的访问权限。安全域的设置限制了用户在数据库中的行为。在这种意义上，安全域包含的信息大约有：

- 用户的认证类型（通过操作系统或网络服务）。
- 用户可用的系统资源的总量，包括表空间（类似于 Windows 或 DOS 环境中的目录）和它们的缺省值
- 用户能存取的数据库对象和在这些对象上能执行的操作。

在用户将系统权限授予其他用户之前，他必须有管理权限。表 8-1 列出了某些基本系统特权。

表 8-1 部分系统特权表

系统特权名	允许用户进行的操作
CREATE SESSION	连接数据库
CREATE TABLE	在自己的模式中创建表，在该表上使用如 ALTER、DROP、TRUNCATE 这样的命令
SELECT [ANY TABLE]	查询在自己的模式中的表，假如授予了 ANY，用户能访问其他模式中的表
CREATE SEQUENCE	在自己的模式中创建一个序列
CREATE VIEW	在自己的模式中创建视图

8.1.2.1 建立用户

为建立一个用户，使用下面简化版本的 CREATE USER 命令。

```
CREATE USER user-name IDENTIFIED BY user-password
DEFAULT TABLESPACE tablespace-name
TEMPORARY TABLESPACE temptablespace-name
QUOTA integer M on tablespace-name
QUOTA integer N on temptablespace-name;
```

这个命令指派给用户一个缺省表空间和临时表空间名。正如以前已经提到过的，表空间是一个逻辑存储单元，类似于 Windows 或 DOS 中的目录。这个命令还确定了分配给这些表

空间中每个用户的内存总量。

正如前面指出的那样，创建一个用户需要有管理特权。为了能做到这一点，需要用 SYSTEM 作为用户名，MANAGER 作为口令，登录进 P08。这个内部账户在 Oracle 中具有管理特权[①]。

例 8.2

创建一个用户名为 hayley，口令是 'whm123bng' 的用户。

建立这个用户使用的命令如下：

```
CREATE USER hayley IDENTIFIED BY whm123bng
DEFAULT TABLESPACE user_data
TEMPORARY TABLESPACE temporary_data
QUOTA 5M on user_data QUOTA 1M on temporary_data;
```

表空间 user _ data 和 temporary _ data 是在 P08 缺省数据库内建立的。在这节中，我们使用它们，不必进一步考虑它们的结构，因为其解释已超出了本书的讨论范围。

8.1.2.2　删除用户

有时，可能需要将用户从数据库中删除。例如，一个雇员辞职或者被解雇了。下面的指令允许我们做这件事情：

DROP USER user-name [**CASCADE**];

在删除用户时若使用 CASCADE 选项，则不仅会删除用户，而且会删除其模式中的用户对象。

例 8.3

删除用户 Hayley，但保留其模式中的所有对象。

```
DROP USER hayley;
```

8.1.2.3　监控用户

为显示有关用户的信息，有几种视图（在本章后面进行解释）允许数据库管理员收集有关数据库用户的信息而且可以像任何一个其他的普通表一样查询这些视图。表 8-2 列出了提供用户信息的一些视图。

表 8-2　提供用户信息的部分视图列表

视图名	可获得的用户信息
ALL _ USERS	已创建的所有用户

① 在以下各节中，除非特别说明，否则我们假设读者用这个内部账户登录进入 Oracle 数据库。

（续表）

视图名	可获得的用户信息
USER_USERS	当前登录的用户
USER_TABLES	当前用户拥有的所有表
USER_TS_QUOTAS	当前用户对表空间的引用
USER_OBJECTS	用户拥有的对象

8.1.2.4 为用户授予系统特权

一旦建立了一个用户，系统管理员就可以授予该用户一组特权。表 8-3 列出了一些授予用户的最常用的特权。

表 8-3 授予用户的常用特权的部分列表

特权名	允许用户进行的操作
CREATE SESSION	连接数据库
CREATE TABLE	在她或他的模式中创建表
CREATE VIEW	在她或他的模式中创建视图
CREATE SEQUENCE	在她或他的模式中创建序列

为用户授权的命令如下：

```
GRANT priv-1 [ , priv-2, … priv-n ] TO USER user-name;
```

授给用户的第一个特权应该是允许用户与该数据库连接，在表 8-3 中，这个特权的名字为 CREATE SESSION。

例 8.4

重建 hayley 用户，并授予他 CREATE SESSION 权限。

对应的命令如下：

```
GRANT create session TO hayley;
```

如果用户 hayley 试着登录进该数据库，登录会成功。若她尝试去建表，则不能达到目的，因为她没有这个特权。

例 8.5

授予用户 hayley 创建数据表和视图所需要的权限。

对应的命令如下：

```
GRANT create table, create view TO hayley;
```

这个语句执行以后，用户 hayley 能连接上数据库，并能在它里面建立数据表和视图。

为了避免为不同用户授予特权这种令人生厌的工作，数据库管理员建立了相关特权组，

并授给那些需要这些特权的用户。每个特权组被称为角色（role）。例如，数据录入操作员可能需要某些特权集合，而程序设计员则需要不同的特权集合。数据库管理员能分别为录入操作员和程序设计员创建角色。然后，DBA 将特权分配给每个角色，并把它们再授权给两个不同的用户组。

创建角色的命令如下：

```
CREATE ROLE role-name;
```

为给已创建的角色授予特权，DBA 可使用如下所示的 GRANT 命令：

```
GRANT privilege-1 [, privilege…] TO role-name;
```

用下面的例子说明这一点。

例 8.6

建立一个角色，允许用户有创建会话和建表的特权。同时建立另外一个角色，允许用户创建会话、视图，但不允许创建表。这些角色分别命名为 role _ tables 和 role _ views。

```
CREATE ROLE role_tables;
GRANT create session, create table to role_tables;
CREATE ROLE role_views;
GRANT create session, create view to role_views;
```

创建这些角色后，DBA 使用 GRANT 命令把它们指派给任一用户。假设用户 Nancy Wendt（wendtnan）和 Denise Fernandez（fernande）需要访问数据库，并且要创建表但不创建视图，DBA 会用如下的命令把这些特权授予他们：

```
GRANT role _ tables TO wendtnan , fernande;
```

将特权授予了用户或角色后，也能从用户或角色中拿走。下面的命令允许 DBA 从用户或角色处收回指定的特权集合：

```
REVOKE priv-1 [,priv-2...priv-n ]
FROM [ user-1 [, user-2...,user-n]]
```

从一个或几个用户处收回各个特权的命令

```
or

REVOKE priv-1 [,priv-2...priv-n ]
FROM [ role-1 [, role-2...,role-n]];
```

从一个或几个角色处收回各个特权的命令

已被授予特权的用户可以通过使用系统提供的有关授权信息的某些视图，查阅用户被授予的特权。表 8-4 列出了这些视图的一部分。

表 8-4　部分提供有关授权信息的视图列表

视图名称	允许用户的操作
USER _ SYS _ PRIVS	检查授予当前用户的系统特权
USER _ TAB _ PRIVS _ RECD	检查那些用户被授予的对象特权
USER _ ROLE _ PRIVS	授予用户的角色

8.1.2.5 为用户授予对象特权

到目前为止，我们考虑了系统特权。然而，DBA 也能为指定对象授予特权。这种类型的特权称为对象特权。对象特权允许对对象采取特定动作。表 8-5 列出了一些对象和它们的特权。

表 8-5 部分表和视图的对象特权列表

OBJECT PRIVILEGE	TABLE	VIEW
ALTER	√	
DELETE	√	√
INSERT	√	√
REFERENCES	√	
RENAME	√	√
SELECT	√	√
UPDATE	√	√

在指定对象上，为用户或角色授予对象特权的命令如下：

```
GRANT obj-priv-1 [,obj-priv-2...,obj-priv-n]
ON object-1 [,object-2,...,object-n]
TO user-1[,user-2,...,user-n]
[WITH GRANT OPTION];
```

or

```
GRANT obj-priv-1 [,obj-priv-2...,obj-priv-n]
ON object-1 [,object-2,...,object-n]
TO role-1 [,role-2,...,role-n]
[WITH GRANT OPTION];
```

若使用了 WITH GRANT OPTION 选项，被授予此特权的用户或角色反过来也能把这些特权转授给其他用户。任何拥有对象的用户都可将特权授予另一位用户。

例 8.7

scott 已登录数据库，写一个命令，以使 scott 把在表 S_EMP 和 S_DEPT 上的 SELECT 特权授予用户 hayley。

```
GRANT select
ON s_emp
TO hayley;
GRANT select
ON s_dept
TO hayley;
```

对象的拥有者能将对象上的特权授给其他用户，用户可以通过将拥有者名放在对象名前面，拥有者名和对象名之间用句号分开的形式引用这些对象。

```
Owner-name.object-name
```

例如，用户 hayley 已登录，她想访问 S_EMP 表，可通过将拥有者名 scott 放在表名前完成。她应该这样引用这个表：

```
SELECT *
                    ——————— 注意，这个句号把对象的拥有者名和对象名分隔开
FROM scott.s_emp;
```

8.2　通过视图隐藏数据

有一种机制允许用户将其他用户不能看到的数据排除在外，这种机制称为视图。用这个术语引用的实际是一个存储查询，该查询执行的时候，从其他表（基表或基础表）导出它的数据。视图的定义逐字地存储在数据字典中，视图不包含数据，也不为它分配存储空间。当用户引用一个视图时，RDBMS 从数据字典中检索它的定义，并执行或运行该查询。

视图提供了安全性，因为它们允许我们将数据隐藏在查询的背后。也就是说，执行查询的用户仅能看到查询的结果，而不是从中提取数据的表或视图。在这种意义上，视图是显示剪裁了的数据，这些数据包含在一个或者多个表中，或者在另外的视图中。注意，视图能建立在某些其他视图之上，因而提供了一种额外的数据保护。

8.2.1　创建视图

由于视图可以从表中导出，因此视图和表这两个对象之间有许多类似的地方。从实用的观点看，用户很难区分表和视图有什么不同。具有相应权限的用户可以像查询任意其他表一样查询视图。另外，用户能更新、插入和从视图中删除数据，只是带有某些限制而已。像任何其他的数据库对象一样，要在自己的模式中创建视图，用户必须有 CREATE VIEW 系统特权。

创建视图命令的语法如下所示。

```
CREATE VIEW view-name [(alias-1, alias-2,...,alias-n)]
AS query
WITH [READ ONLY | WITH CHECK OPTION [CONSTRAINT constraint-name];
```

这些别名是视图的查询所选择的表达式的命名。别名的个数必须与视图所选择的表达式个数相匹配。WITH READ OPTION 选项防止通过视图来插入、删除或更新基础表。WITH CHECK OPTION 选项指明通过视图执行的插入、修改操作完成后，会产生视图查询可以选

择的行[①]。下面的例子说明了这个问题。

例 8.8

创建一个视图，包含 s_customer 表（基表）的如下属性：name，city，state，country。视图名为 European_client。

```
CREATE view European_client
AS SELECT name, city, country, region_id
FROM s_customer
WHERE region_id = '5'
WITH CHECK OPTION;
```

为了显示这个视图的内容，用户可发出如下命令：

```
SELECT *
FROM european_client;
```

这个查询的结果如下：

```
NAME                 CITY                 COUNTRY              REG
-------------------- -------------------- -------------------- ---
Sportique            Cannes               France               5
Muench Sports        Munich               Germany              5
Helmut's Sports      Prague               Czechoslovakia       5
Sports Russia        Saint Petersburg     Russia               5
```

例 8.9

创建一个名为 sales person 的视图，显示运动用品数据库中所有销售代表的姓氏和部门名称。标题名为 Employee 和 Department。

允许我们显示该信息的这个查询是表 S_EMP 和表 S_DEPT 在共同属性 dept_id（S_EMP 表的属性）和 id（S_DEPT 表的属性）上的连接。相应的命令如下：

```
CREATE VIEW Sales_Person (Employee,Department)
AS SELECT E.last_name, D.name
FROM s_emp E, s_dept D
WHERE E.dept_id = D.id
AND E.title ='Sales Representative';
```

这个视图执行的输出结果如下：

① 实际上，若视图或该视图基于的任何基础视图的查询中包括子查询，则这个 WITH CHECK OPTION 选项不能做这个担保。

```
SELECT *
FROM SALES_PERSON;

EMPLOYEE             DEPARTMENT
-------------------- ----------
Henderson            Sales
Gilson               Sales
Sanders              Sales
Dameron              Sales
```

这个例子清楚地说明了视图有效地隐藏了数据。对用户来说，该视图和其他表之间没有什么实际的不同。事实上，假如描述一下这个视图（见下面），我们根本不会意识到它的基表是两个表的连接。此外，我们还可以观察到别名怎样更好地帮助隐藏了数据，因为根本就没有提及原始列的真实列名。

```
DESC sales_person;
 Name                              Null?    Type
 --------------------------------- -------- ----
 EMPLOYEE                          NOT NULL VARCHAR2(20)
 DEPARTMENT                        NOT NULL VARCHAR2(20)
```

8.2.2　更新视图

正如前面指出的，用户假如拥有适当的特权，可以更新视图。更新视图的命令与更新表的命令类似（参见 1.16.1 节）。然而在某些情况下，为了保护数据的完整性，需要确保任何通过视图完成的更新都不会影响该视图能够查询的数据。通过在创建视图时带 WITH CHECK 选项就可以制止这种类型的更新，正如在例 8.8 中那样。下面的例子说明了这一点。

例 8.10

更新例 8.8 中的查询，将位于俄罗斯的所有客户的地区标志 region-id 从原来的 5 改为 2。

对应的命令如下：

```
UPDATE European_client
SET region_id ='2'
WHERE country = 'Russia';
```

注意，假如尝试着执行这个命令，会得到一个错误。这个错误如下：

```
UPDATE European_client
SET region_id ='2'
WHERE country = 'Russia';
UPDATE European_client
       *
ERROR at line 1:
ORA-DDD: view WITH CHECK OPTION where-clause violation.
```

在这种情况下，DDD 表示制造了一个内部错误。当我们试图改变该查询检索的某一行时出现了这一错误。也就是说，如果我们更新了这行，该查询将不能再检索到它，因为它已将 region _ id 从 5 改为 2 了。在创建视图时，如果不用 WITH CHECK OPTION 选项，我们就能够更新视图和它的基础表。

问题与答案

8.1 写一个命令，将用户 Jose Luis Fernandez（fernajol）的口令由原来的 1plummer82 改为 3wnatasha2。

为了改变该用户的口令，需要发出下面的命令：

```
ALTER USER fernajol IDENTIFIED BY 3wnatasha2;
```

8.2 创建一个用户名为 acm123mca、口令为 bnd26cufm 的用户。使用 P08 提供的 user _ data 和 temporary _ data 表空间，并在 user _ data 中给这个用户提供 10M 的存储空间，在 temporary _ data 中提供 5M 的存储空间。

使用 P08 提供的表空间创建该用户的命令如下：

```
CREATE USER acm123mca IDENTIFIED BY bnd26cufm
DEFAULT TABLESPACE user_data
TEMPORARY TABLESPACE temporary_data
QUOTA 10M on user_data QUOTA 5M on temporary_data;
```

8.3 创建名为 role _ tables _ and _ views 的角色。

创建这个角色的命令如下：

```
CREATE ROLE role _ tables  and _ views;
```

8.4 为上面问题中创建的角色授予连接数据库的特权以及创建表和视图的特权。

连接数据库的特权是 CERATE SESSION 。创建表和视图的特权分别是 CREATE TABLE 和 CREATE VIEW。

为指定的角色授予这些特权的命令如下：

```
GRANT create session, create table, create view
TO role _ tables _ and _ views;
```

8.5 将上面问题中的角色授予用户 fernajol 和 acm123mca。将角色授予这两个用户以后，他们能做什么？

```
GRANT role_tables  and_views TO fernajol , acm123mca;
```

两个用户都能与数据库连接，并能创建表和视图。

8.6　用户 fernajol 和 acm123mca 没有在表 S_INVENTORY 和表 S_ITEM 上的 SELECT 特权，这些表是 scott 建立的。写一个命令，让 scott 将两个表的 SELECT 特权授予他们。

用户 scott 必须发出如下命令，允许这两个用户在指定的表上进行查询：

```
GRANT select ON s_inventory TO fernajol, acm123mca;
                              and
GRANT select ON s_item TO fernajol, acm123mca;
```

8.7　用户 fernajol 已经调职，不再需要通过角色 role_tables_and_views 授予的特权。确保他不再能创建表和视图或者有访问上题表的特权。然而，用户 fernajol 仍保留连接数据库的权力。写一个命令完成这些任务。

下面的命令从用户 fernajol 处收回了上面提到的那些特权：

```
REVOKE select ON scott.s_inventory FROM fernajol;
REVOKE select ON scott.s_item FROM fernajol;
REVOKE create table, create view FROM fernajol;
```

8.8　假定用户 fernajol 结束了他所有的工作，并且已经到了别的公司。以前他创建的对象已不再有任何用处。写一个命令，允许 DBA 删除该用户和他的所有对象。

相应的命令如下：

```
DROP USER fernajol CASCADE;
```

注意，删除用户和他的对象，CASCADE 选项是必需的。

8.9　假如 DBA 怀疑当前已登录的一个人是合法用户 dulldns 的冒充者。有什么办法终止这个人与数据库的连接？

是的，有办法。然而，在结束会话之前，DBA 需要获得一些信息，比如会话 id 和用户会话的序列号。DBA 能从 V $SESSION 视图中得到这些信息。下面是能检索该信息的一个查询以及输出的结果：

```
SELECT sid,serial#,username FROM v$session;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         1          1 dulldns
         2          3 scott
         .
         .
         9         11 SYSTEM
```

根据这些信息，DBA 可以通过发出下面的命令取消这次会话：

```
ALTER SYSTEM KILL SESSION '1, 1';
```

8.10　作为一个用户，如何找出指派给我的缺省和临时的表空间？

数据字典视图 USER_USERS 允许任何用户找到这一信息。另外，除了以前的信息

外，该视图还能通知用户有关他的用户 ID 和创建账户的时间。

补 充 题

8.11 将用户 Danielle Armstrong（armstdan）的口令由原来的 burg8two 改为 1span2nts。

8.12 我们如何删除一个视图?

8.13 创建一个用户名为 gomezlila，口令为 titus2 的用户。使用 P08 提供的表空间 user _ data 为该用户在 user _ data 中分配 12M 的存储，在 temporary _ data 中分配 6M 的存储。

8.14 创建角色 role _ tables。

8.15 为前一问题中创建的角色授权连接数据库和创建表的特权。

8.16 将前一问题中的角色授权给用户 math7gnm 和 halmat14。

8.17 允许用户 math7gnm 和 halmat14 在由用户 scott 建立的表 S _ WAREHOUSE 和 S REGION 中有 SELECT 特权。

8.18 从用户 math7gnm 处，收回他的所有特权。

8.19 删除用户 math7gnm 以及他在数据库中的所有对象。

8.20 可以删除当前与数据库连接的用户吗?

补充题答案

8.11

```
ALTER user armstdan IDENTIFIED BY 1span2nts;
```

8.12 像大多数数据库对象一样，可以使用 DROP 命令删除视图。这个命令的语法与删除表的命令的语法类似。例如，为删除视图 european _ client，我们可以发出如下的命令：

```
DROP VIEW european _ client;
```

注意，在这儿用 CASCADE 选项没有意义，因为在视图上不可能定义完整性约束条件。

8.13

```
CREATE USER gomezlila IDENTIFIED BY titus2
DEFAULT TABLESPACE user_data
TEMPORARY TABLESPACE temporary_data
QUOTA 12M on user_data QUOTA 6M on temporary_data;
```

8.14

```
CREATE ROLE role_tables;
```

8.15

```
GRANT create session, create table
TO role_tables;
```

8.16

```
GRANT role_tables TO math7gnm, halmat14;
```

8.17

```
GRANT select ON s_inventory TO math7gnm, halmat14;
                              and
GRANT select ON s_item TO math7gnm, halmat14;
```

8.18

```
REVOKE create session FROM math7gnm;
REVOKE create table FROM math7gnm;
REVOKE select scott.s_warehouse FROM math7gnm;
REVOKE select scott.s_item FROM math7gnm;
```

8.19

```
DROP USER math7gnm CASCADE;
```

8.20 不可以。不能删除当前与数据库连接的用户。然而，DBA 能终止他或她的会话，然后再删除该用户。

附录 A　Personal Oracle 的使用

什么是 Personal Oracle

Personal Oracle 是 Oracle 8 数据库的 PC 版。它是一个完整的关系数据库管理系统，且与一组 Oracle 数据库管理工具（如 Personal Oracle 8 Navigator for Windows 95、Oracle 备份和恢复工具、Oracle 实用程序、Oracle Objects for OLE，Oracle 8 ODBC 驱动程序和 Oracle 联机文档等）捆绑在一起。

从什么地方可以得到 Personal Oracle

可以从 Oracle 的网站 http://www.oracle.com 下载 90 天免费试用版的拷贝，或者从同一地方交少许费用订购一个试用版拷贝。

主要组件

虽然 Personal Oracle 有一系列管理工具，但我们只能使用 SQL 和 Oracle 公司的专有界面 SQL * Plus 创建、访问和操纵 Oracle 数据库。

启动 Personal Oracle

1. 单击 Windows 任务栏上的“开始”按钮。
2. 单击 Oracle 8 Personal Edition，在子菜单上单击 Start Database。假如需要输入一个口令，请用“ORACLE”单词作为口令。
3. 单击“程序”按钮。
4. 单击 Oracle for Windows NT。
5. 单击 SQL * Plus 8.0，将看到图 A-1 所示的登录屏幕。
6. 输入用户名“SCOTT”和口令“TIGER”。不要在 Host String 字段输入任何内容。这样你就可以使用内部建立的 Scott/Tiger 账户登录。然而，假如用此账户遇到了困难，可以尝试使用“SYSTEM”作为用户名，“MANAGER”作为口令登录。
   注意：可以将两步输入组合成一步，在 User Name 文本框中，输入“scott/tiger”或者“system/manager”即可。
7. 如果登录成功，就会出现图 A-2 所示的屏幕。注意，所展示的仅仅是屏幕的一部分。
8. 现在，可以准备用 SQL * Plus 开始工作了。在这个附录中，我们将用 SQL > 作为 sequel 的提示符。

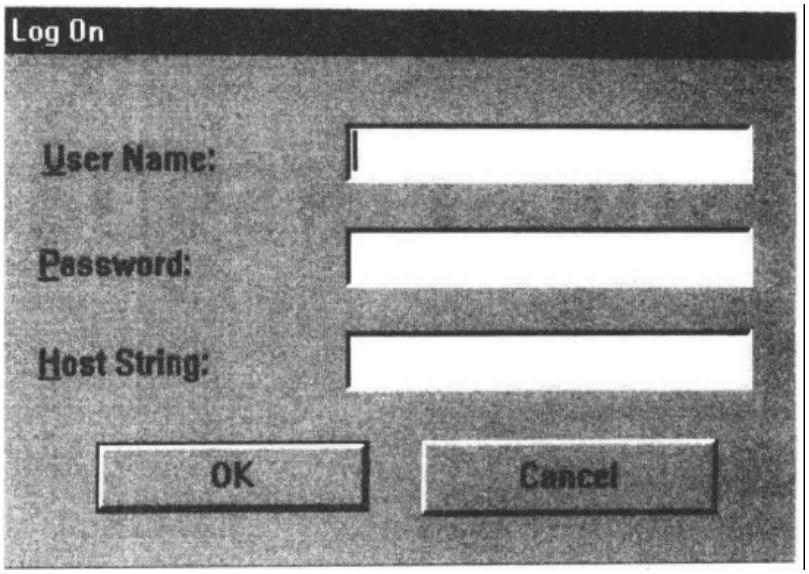


图 A-1　登录屏幕

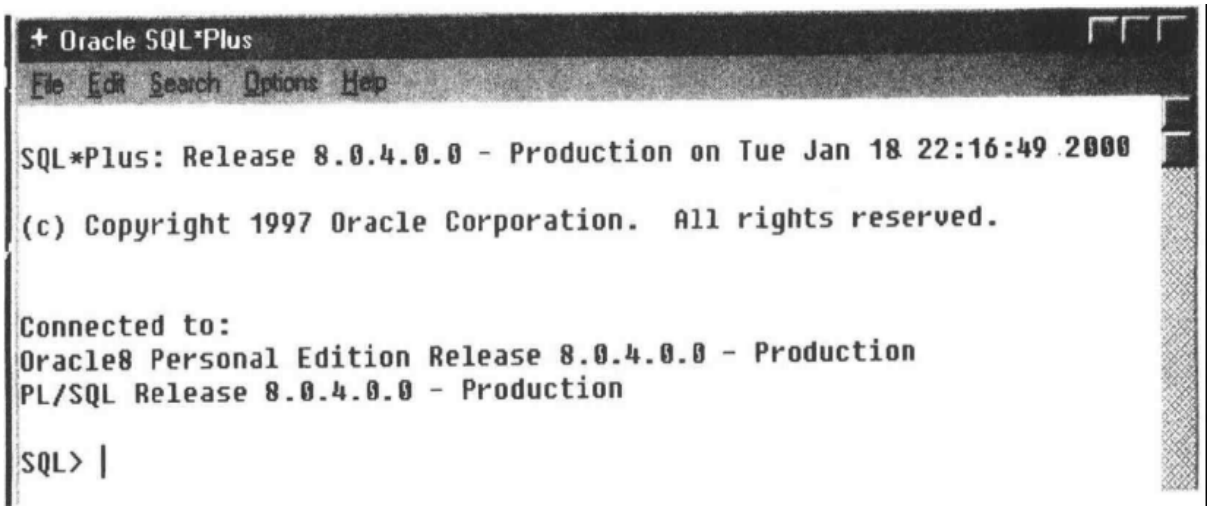


图 A-2　登录成功后的屏幕

9. 只要键入 EXIT，可以在任何时候退出 SQL * Plus。

SQL * Plus 中的编辑器是行编辑器。也就是说，你只能编辑“当前”行。在 SQL * Plus 中，当前行用 * 号标识出来。用户键入的指令存储在临时存储区或缓冲区内。这个缓冲区仅仅保存一条完整的指令。任何时候，要显示缓冲区的内容，用户需要在 SQL > 提示符后键入 list 或 l。为执行缓冲区内的指令，在 SQL > 提示符后，键入一个正斜杠（即/），再按 Enter 键。下一节我们说明用这个编辑器能完成的一些基本功能。

在缓冲区内增加一新行

为把一新行增加到当前缓冲区，首先需要在缓冲区的当前位置插入一个新行。在SQL > 提示符后键入 i 或 I，再按 Enter 键，就可以将行插入到缓冲区中，这使用户处于“输入”状态。可以连续插入多行，如果想退出“输入”状态时，可以在一行中按两次 Enter 键。读

者应该注意到，新行总是插到当前行的后面。为了使缓冲区的某一行成为当前行，用户只需在 SQL > 提示符后键入相应的行号即可。缓冲区中的当前行总是在其行前有一个 * 号标志。

本节下面所示的代码中，假定需要在 FROM emp 子句后添加 **WHERE sal > 1000** 子句：

```
SQL > list
1  SELECT empno, ename
2* FROM emp
```

由于当前行是最后一行，所以不需要输入行号。为插入新行，键入 i 或 I，系统会作出如下响应：

```
SQL > list
1  SELECT empno, ename
2* FROM emp
SQL > i
3
```

输入丢失的子句并按 Enter 键后，系统将添加一行，如下所示。再按 Enter 键，退出插入状态。

```
SQL > list
1  SELECT empno, ename
2* FROM emp
SQL > i
3    WHERE sal > 1000
4
SQL >
```

为检验已输入的新行的正确性，键入 L，显示该缓冲区的当前内容。

在当前行修改字符

为在当前行修改一个字符或字符序列，在 SQL > 提示符后使用 Change 命令。这个命令可采用下列的任一种格式：

```
Change /old-value/new-value/
```

或

```
C /old-value/new-value/
```

这里，old-value 是在当前行需要修改的字符或字符序列，new-value 是代替 old-value 的新值。下面的例子说明了这个命令的用法。

假定 empno 属性在命令中拼写错误，如下所示：

```
SQL > list
1  SELECT emno, ename  <——— 该行有一个错误
2* FROM emp  <——— 当前行
SQL >
```

因为错误出现在缓冲区的行号 1，所以需要先使其成为缓冲区的当前行。为此，在 SQL > 提示符后，键入 1，并按 Enter 键。改正这个错误的命令序列如下：

```
SQL > list
1  SELECT emno, ename ←——— 需要使其成为当前行
2* FROM emp
SQL > 1
1* SELECT emno, ename ←——— *号表示这行是当前行
SQL > c /m/mp/ ←——— 该命令将第一次出现的字母 m 用 mp 代替

1 * SELECT empno, ename
SQL > /←——— 通常用 / 命令执行当前存储在缓冲区内的命令
```

CHANGE 命令用字母 mp 代替第一次出现的字母 m 。用户键入 L 或 l，显示整个缓冲区的内容；然后，在 SQL > 提示符后键入 / ，执行 SQL 命令。

在行末端追加字符

在当前行末端追加字符或字符序列，在 SQL > 提示符后，使用 APPEND 命令。命令语法如下：

APPEND character-sequence

或

A character-sequence

假定需要在前面例子的 SELECT 子句中添加一个 job 属性，若行号 1 已经是当前行，则完成这一任务的 APPEND 命令如下：

```
1 * SELECT empno, ename

SQL > A , job ←——  APPEND 命令

1* SELECT empno, ename, job ←——— job 属性被加到当前行的末端
```

不习惯用行编辑器工作的用户，可以用像 Notepad 这样的工具编写 SQL 命令，然后，拷贝和粘贴到 SQL * Plus 中。另一种方法是将用户选择的编辑器作为缺省编辑器。为此，发出如下的命令通知 SQL * Plus。

DEFINE _ EDITOR = full-path-to-editor

假如用户要使用 Notepad 编辑器，那么，应该发出如下命令：

DEFINE _ EDITOR = NOTEPAD

任何时候要想使用这个编辑器，用户只要在 SQL > 提示符后键入 EDIT 即可。用户能够从 SQL * Plus 环境将命令拷贝和粘贴到 NOTEPAD 中，反之亦然。

控制会话环境

可以通过设置适当的系统变量来控制 SQL * Plus 环境的行为。可使用如下语句来设置

任何系统变量：

```
SET system-variable = new-value
```

表 A-1 列出了部分最常用的系统变量。用户在 SQL > 提示符后键入 SHOW ALL，可以看到系统变量的完整列表。为了看到某个系统变量的设置，用户可以用如下命令：

```
SHOW system-variable
```

表 A-1 部分系统变量列表

系统变量	说明	示例
FEEDBACK n	当查询或报告返回最小行数 n 时通知用户	SET FEEDBACK 10
	该变量能设置为 ON 或 OFF，缺省值为 6	最小 10 行时通知用户
PAUSE text	查询或报表输出满屏后，暂停时显示的文本	SET PAUSE continue
PAUSE ON|OFF	如果有文本，PAUSE 则显示，使两页之间显示暂停	SET PAUSE ON
PAGESIZE	决定每页的行数。缺省值是 80	SET PAGE 24
LINESIZE	决定每行的字符数	SET LINESIZE 60
UNDERLINE	列标题与列值分开的缺省字符	SET UNDERLINE '*'
		用 * 分开列标题和列值

执行脚本文件

包含 SQL 或 SQL 与 SQL * Plus 组合的文本文件称为脚本文件。为执行 A 驱动器上脚本文件 SG.SQL 中的语句，可使用以下两种命令格式之一：

```
START "a:SG.sql"
```

或

```
@"a:SG.sql"
```

将缓冲区的内容存储到文件中

为了将缓冲区中的内容存储到一个文件中，使用以下命令：

```
SAVE filename [REPLACE]
```

采用选项 REPLACE 替代现有文件中的内容。若该文件不存在，则此选项创建一个新的文件名。若用户不指定文件名的扩展名，则 Oracle 创建的文件带有 SQL 扩展名。

将文件装入缓冲区

要将文件装入缓冲区，用如下命令：

```
GET full-path-to-file filename
```

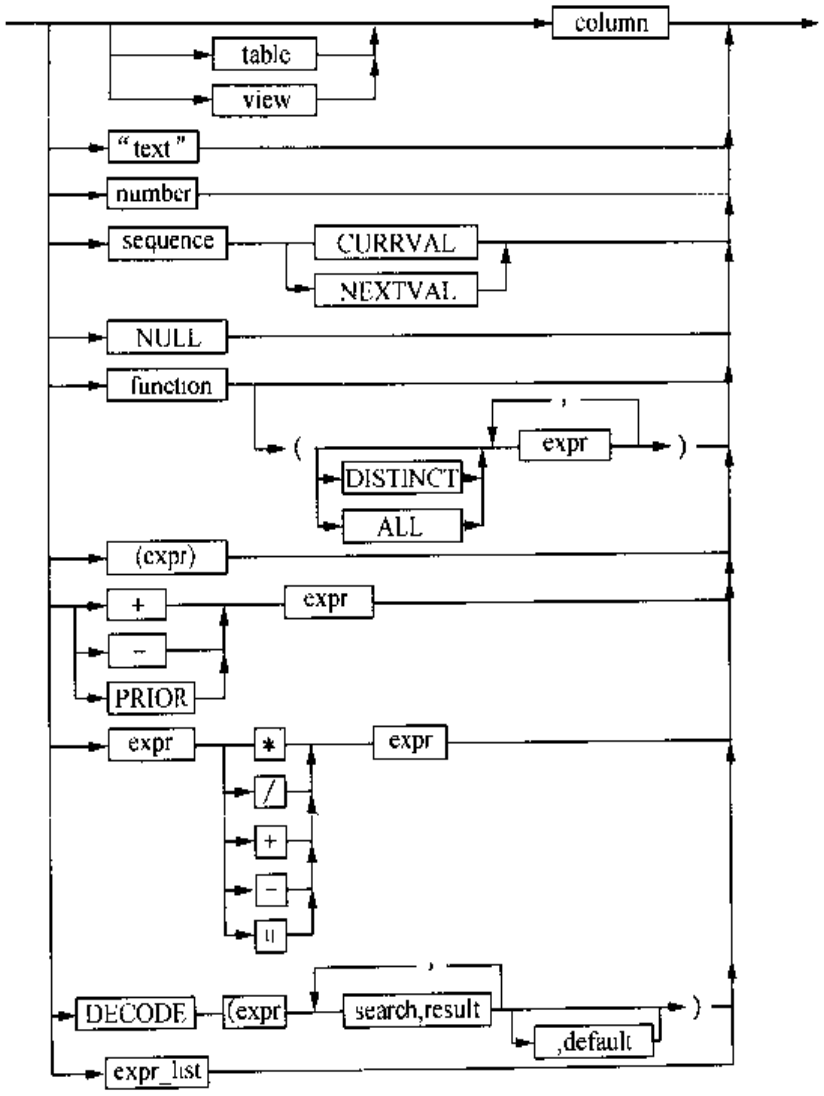
附录 B　SQL 保留字

ACCESS	DEFAULT	INTEGER	OPTION	START
ADD	DELETE	INTERSECT	OR	SUCCESSFUL
ALL	DESC	INTO	ORDER	SYNONYM
ALTER	DISTINCT	IS	PCTFREE	SYSDATE
AND	DROP	LEVEL	PRIOR	TABLE
ANY	ELSE	LIKE	PRIVILEGES	THEN
AS	EXCLUSIVE	LOCK	PUBLIC	TO
ASC	EXISTS	LONG	RAW	TRIGGER
AUDIT	FILE	MAXEXTENTS	RENAME	UID
BETWEEN	FLOAT	MINUS	RESOURCE	UNION
BY	FOR	MODE	REVOKE	UNIQUE
CHAR	FROM	MODIFY	ROW	UPDATE
CHECK	GRANT	NOAUDIT	ROWID	USER
CLUSTER	GROUP	NOCOMPRESS	ROWLABEL	VALIDATE
COLUMN	HAVING	NOT	ROWNUM	VALUES
COMMENT	IDENTIFIED	NOWAIT	ROWS	VARCHAR
COMPRESS	IMMEDIATE	NULL	SELECT	VARCHAR2
CONNECT	IN	NUMBER	SESSION	VIEW
CREATE	INCREMENT	OF	SET	WHENEVER
CURRENT	INDEX	OFFLINE	SHARE	WHERE
DATE	INITIAL	ON	SIZE	WITH
DECIMAL	INSERT	ONLINE	SMALLINT	

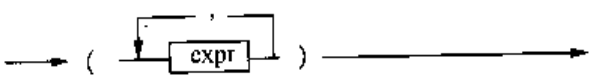
附录 C　SQL 子集的语法图

expr :: =

在下面的语法图中，表达式可能用“expr”表示。

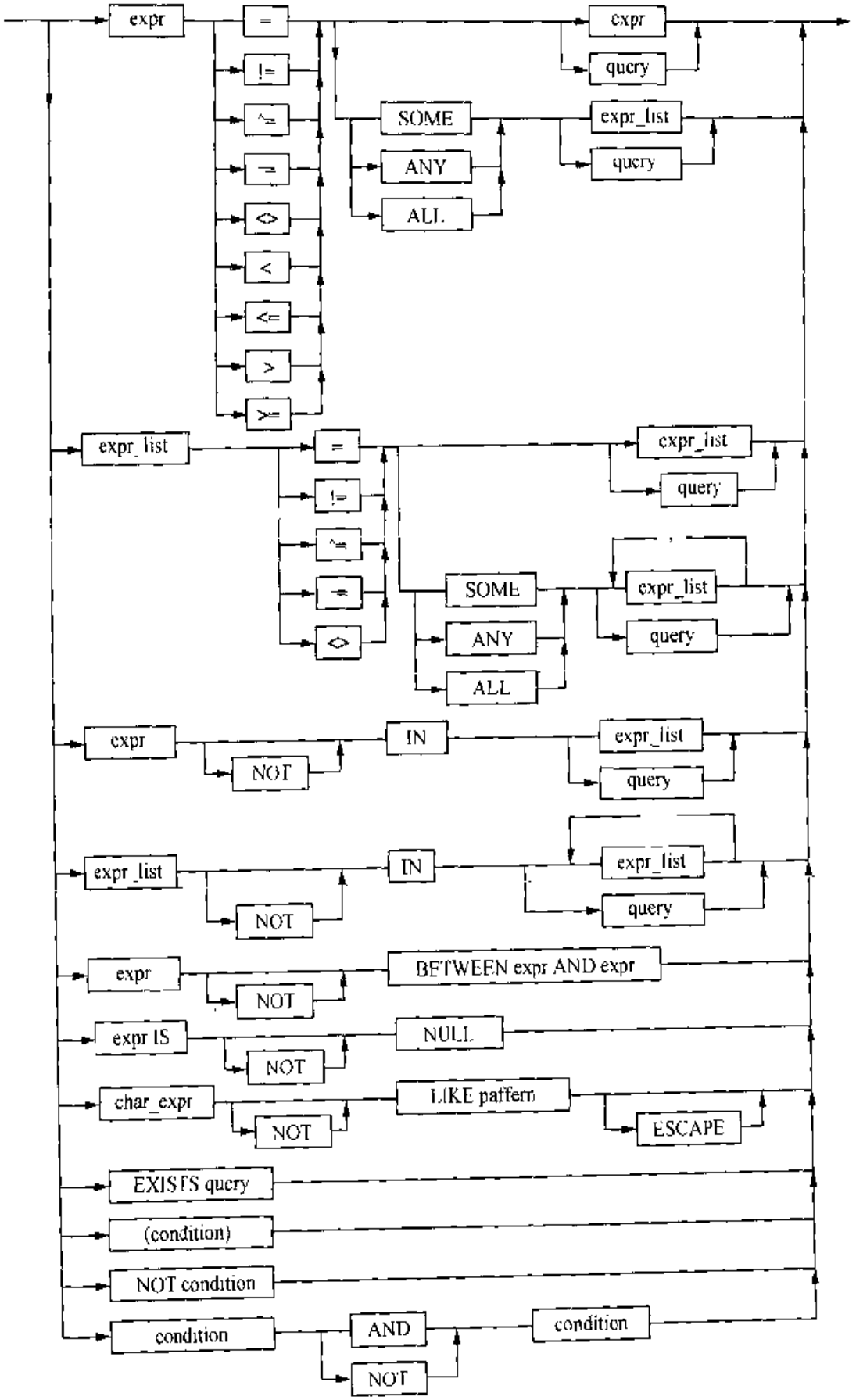


expr _ list :: =



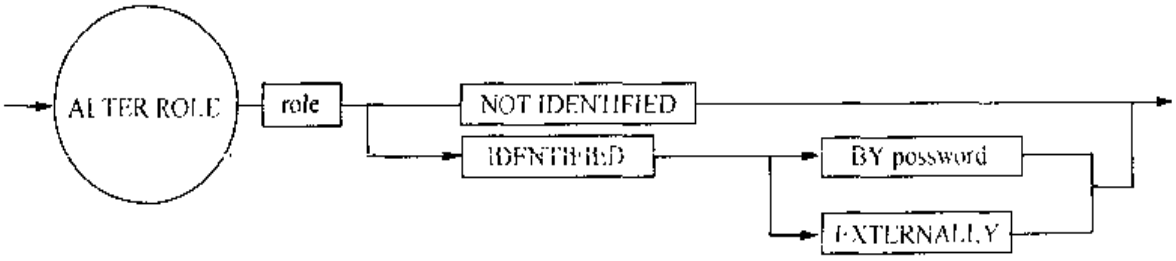
condition :: =

在下面的语法图中，条件可能用“condition”表示。



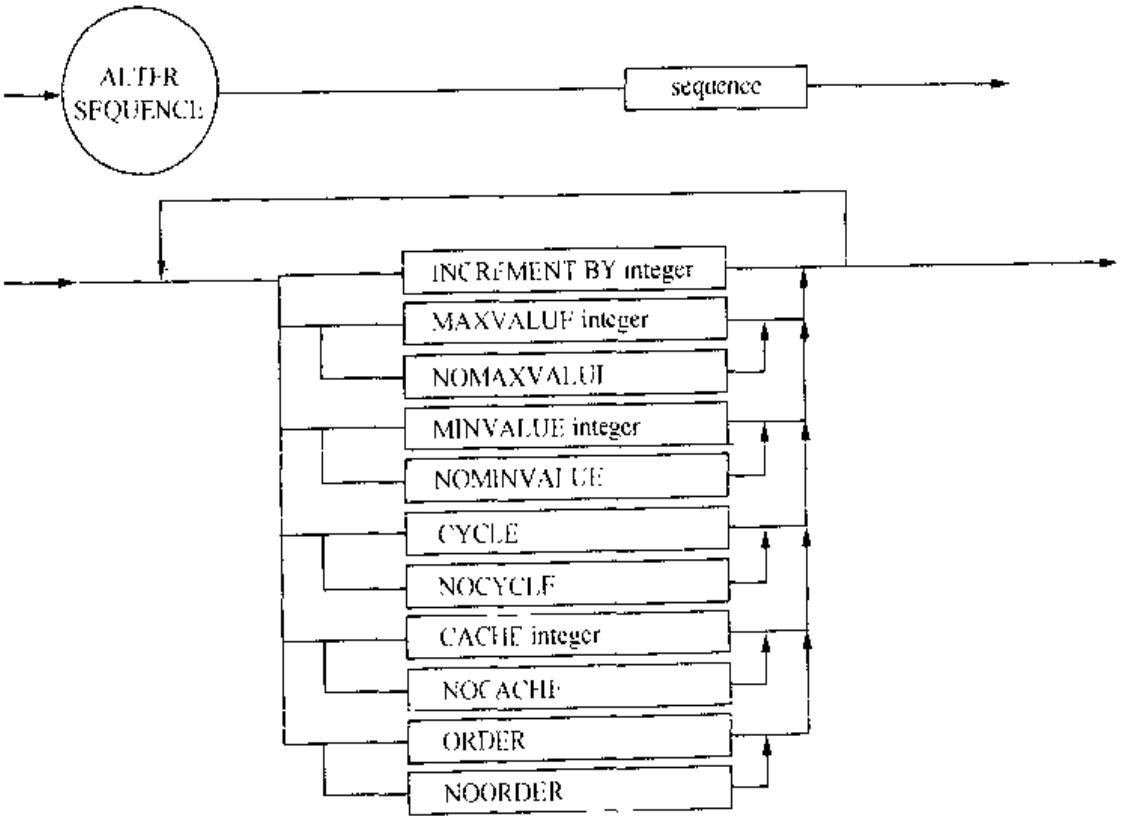
ALTER ROLE（修改角色）

改变支持某一角色需要的授权。



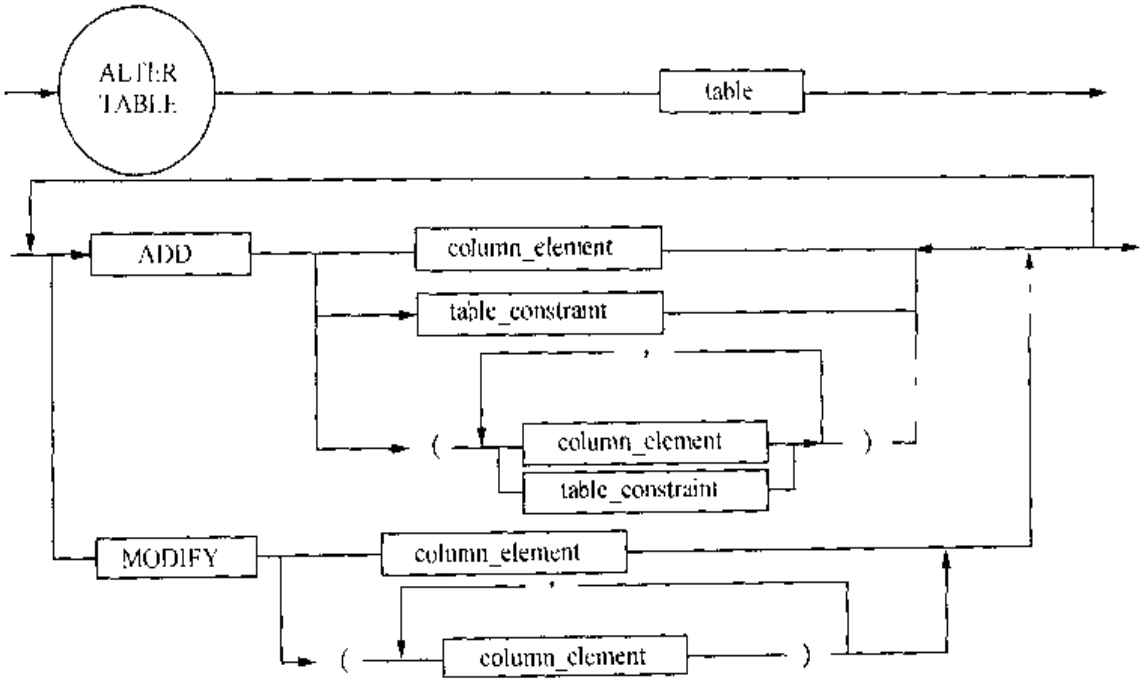
ALTER SEQUENCE（修改序列）

重新定义已存在的序列。

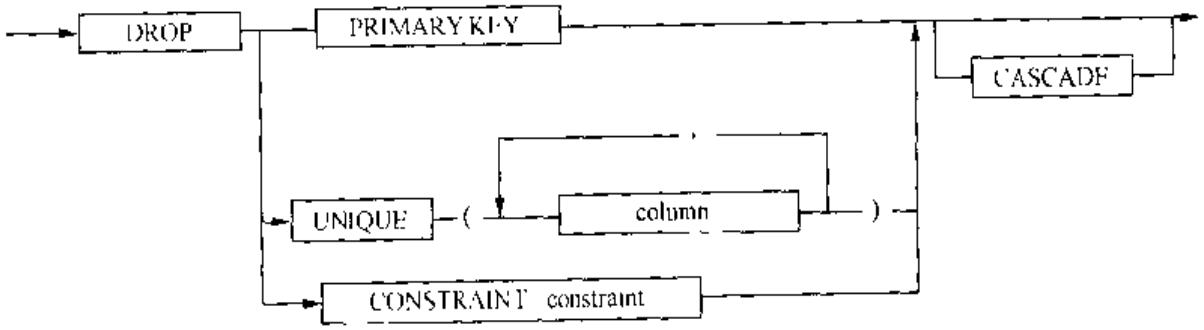


ALTER TABLE（修改表）

重新定义已存在表中的列名、约束或存储分配。

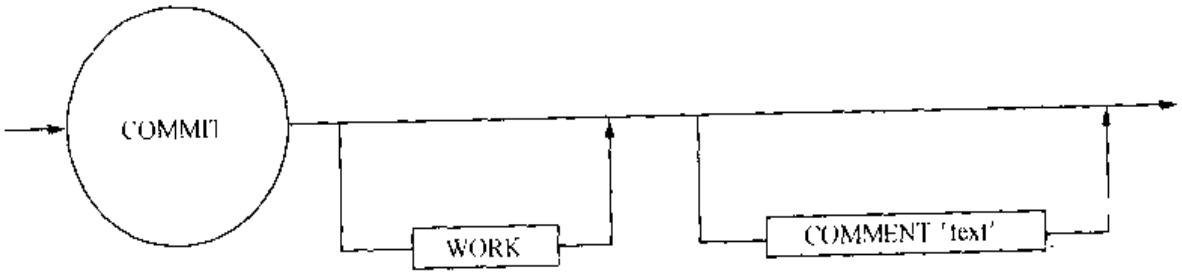


drop_clause::=



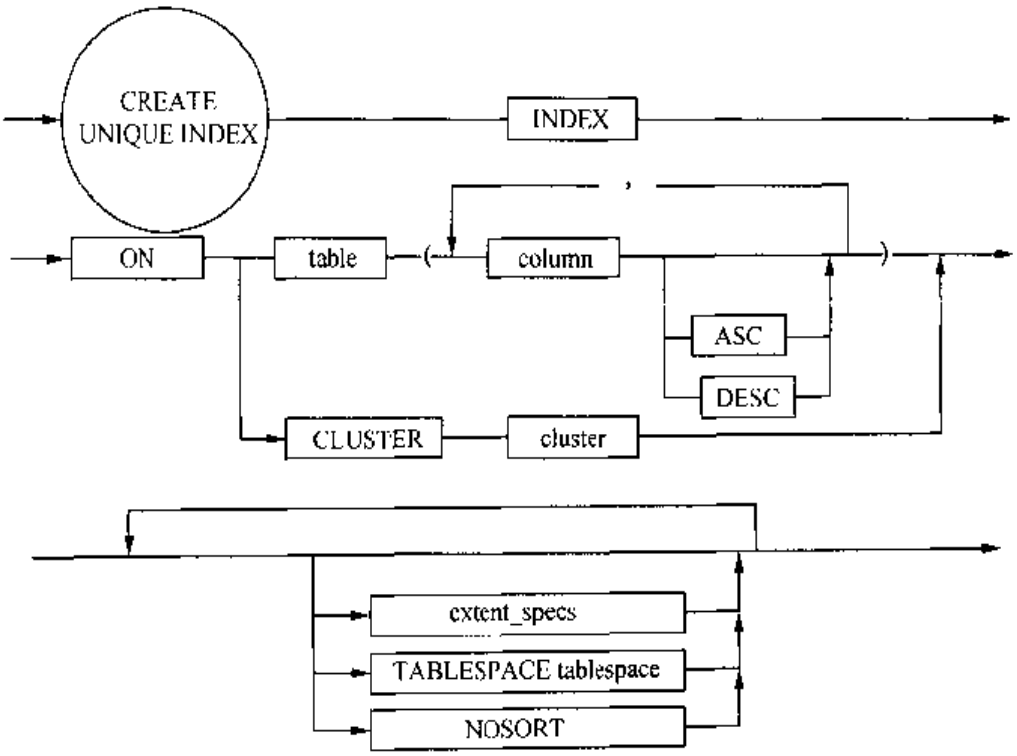
COMMIT（提交事务）

提交事务是保存当前事务期间对数据库的改变、删除事务的保留点并释放该事务占用的全部锁。



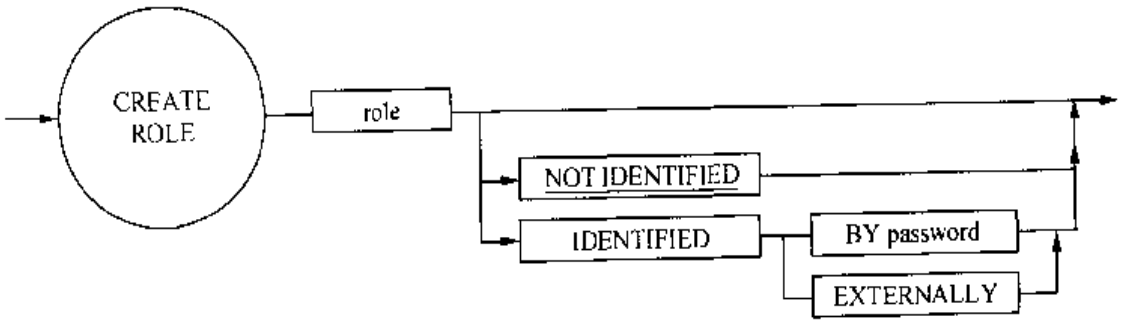
CREATE INDEX（建立索引）

在一个表或一个聚集中，按指定的列建立新索引



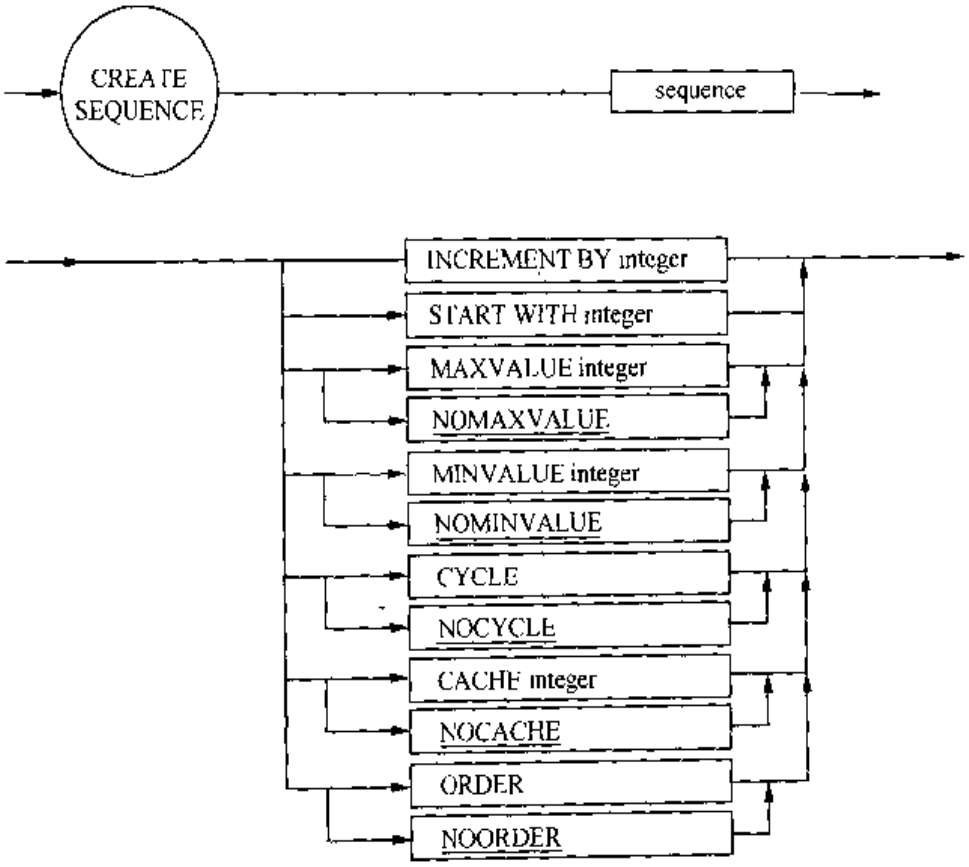
CREATE ROLE（建立角色）

建立一个安全角色。



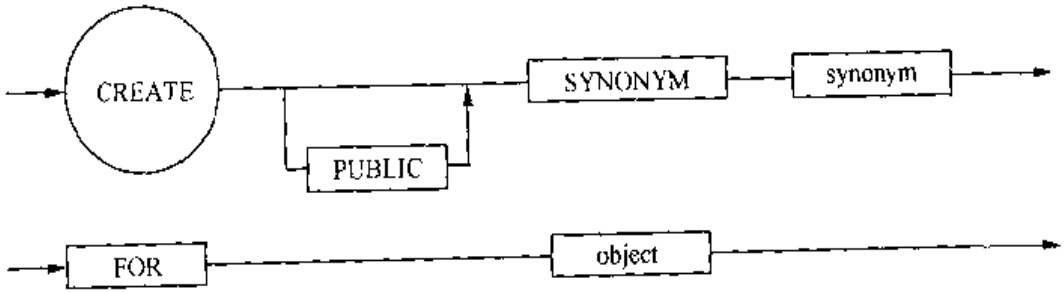
CREATE SEQUENCE（建立序列）

建立一新的序列，以生成主码值。



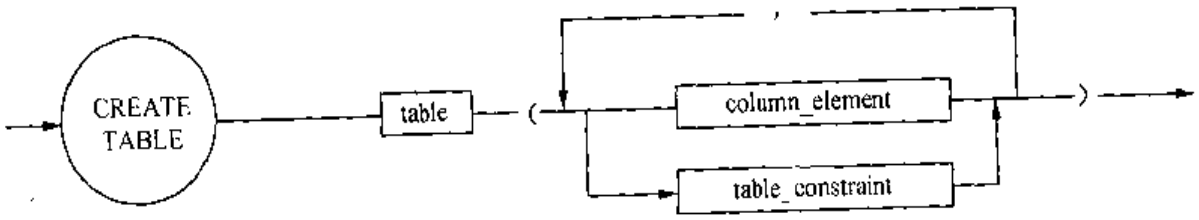
CREATE SYNONYM（建立同义词）

为表、视图、序列或另一同义词创建一个新的同义词。

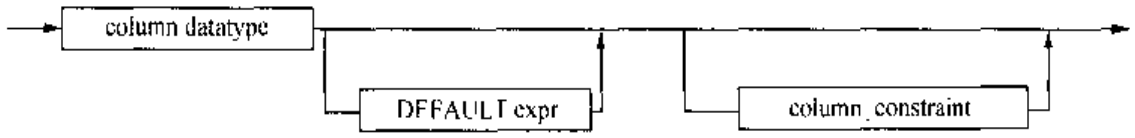


CREATE TABLE（建立新表）

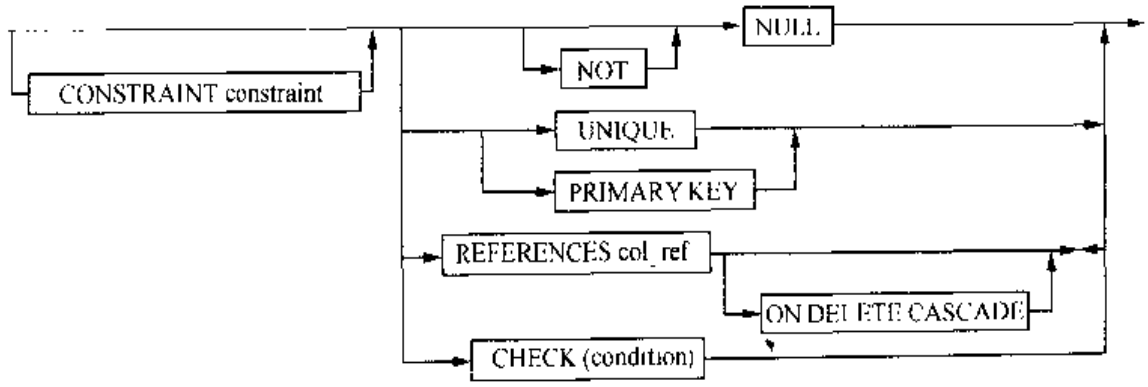
创建一新表，定义它的列名、约束和存储空间。

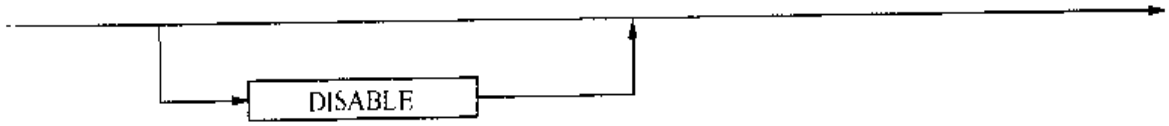


column_element ::=

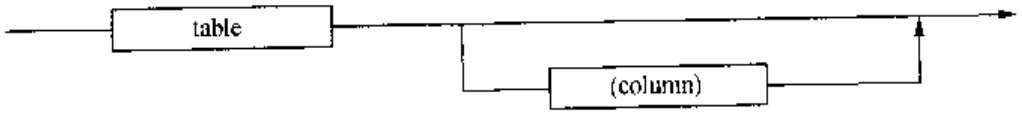


column_constraint ::=

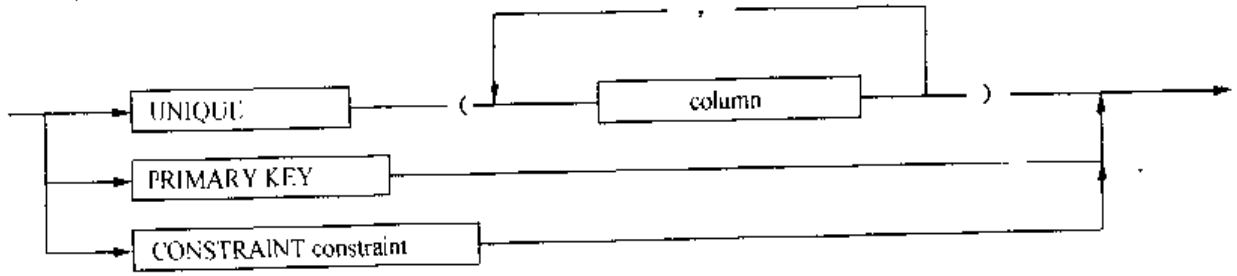




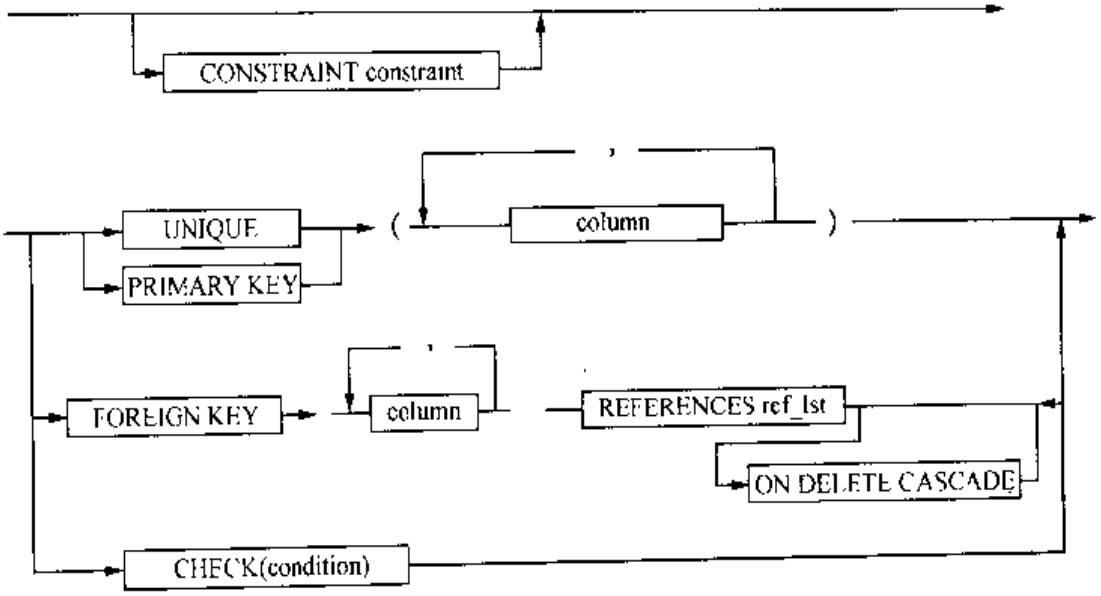
col_ref ::=



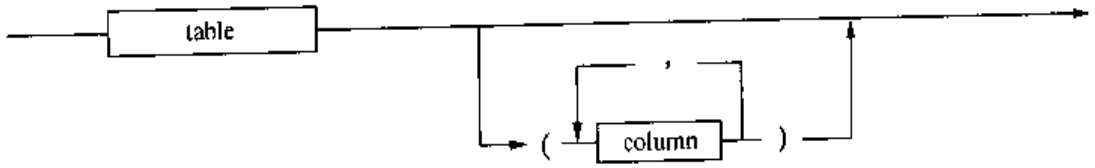
integrity_constraint ::=



table_constraint :: =

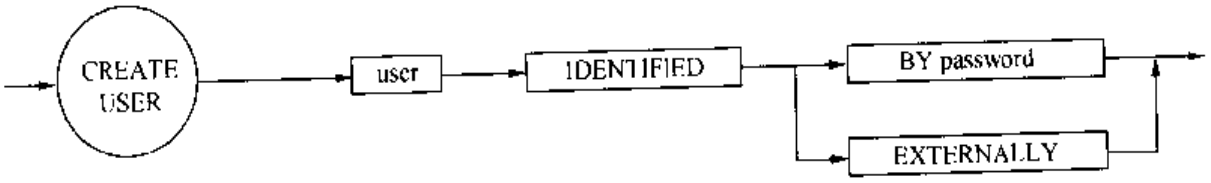


ref_lst :: =



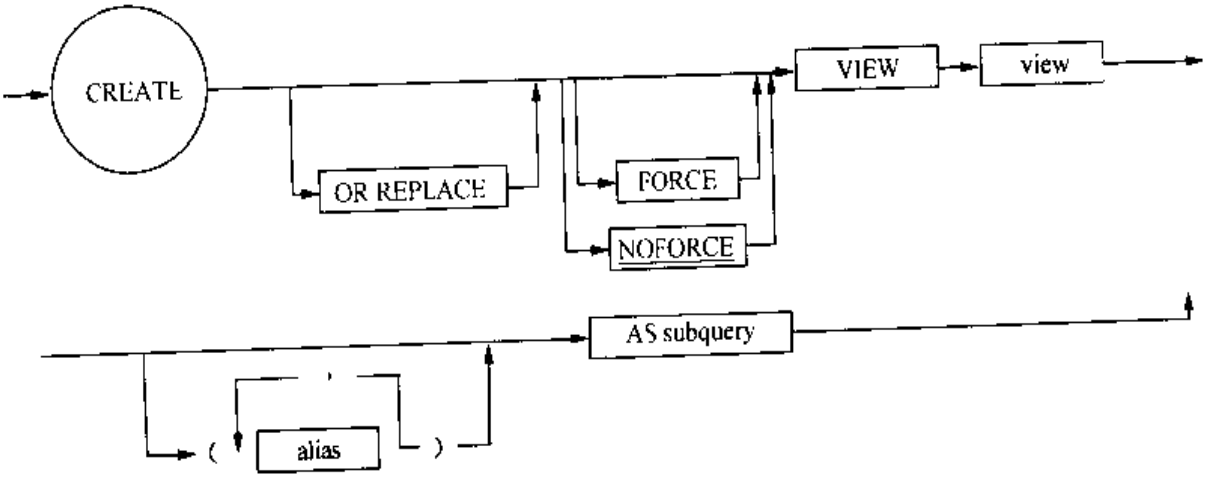
CREATE USER（建立新用户）

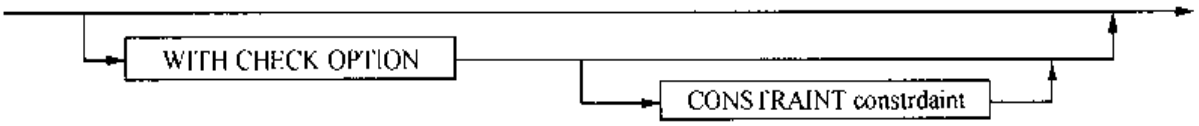
定义一个数据库用户。



CREATE VIEW（建立视图）

建立一个或多个表和/或其他视图的一个新视图。

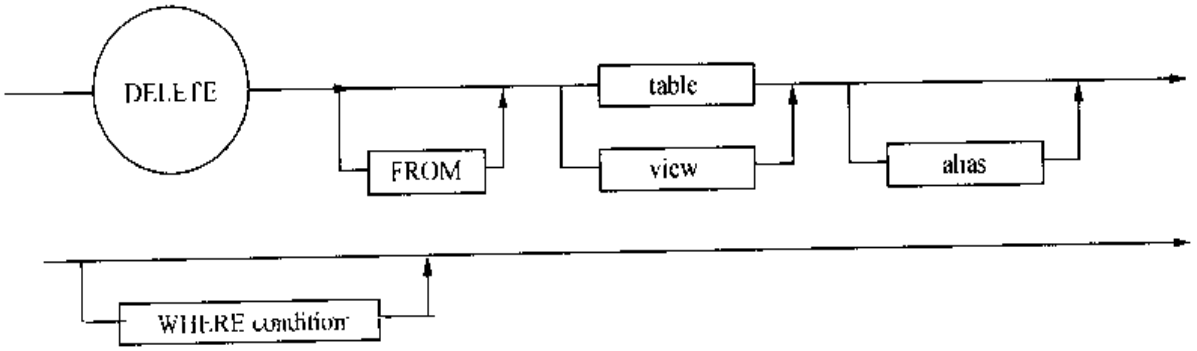




DELETE（删除行）

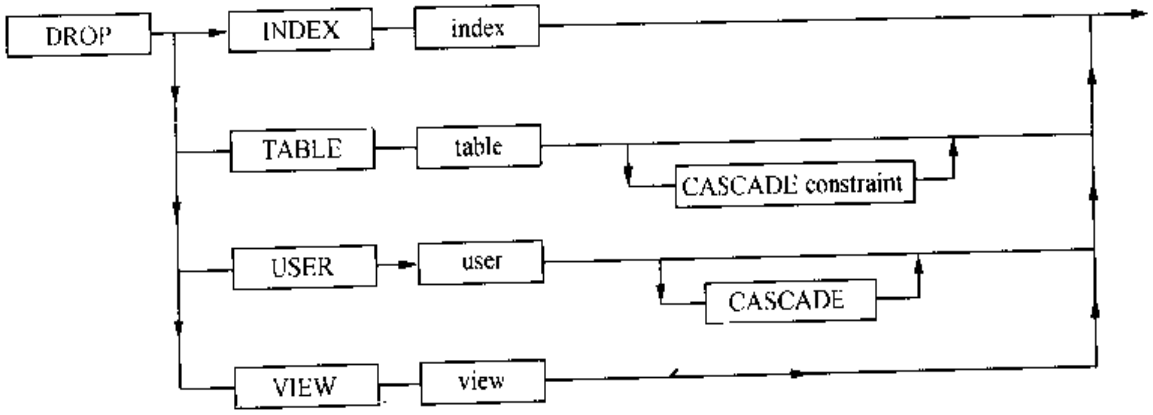
从满足 WHERE 条件的表或视图中删除若干行。

若未指定 WHERE 子句，则删除全部行。



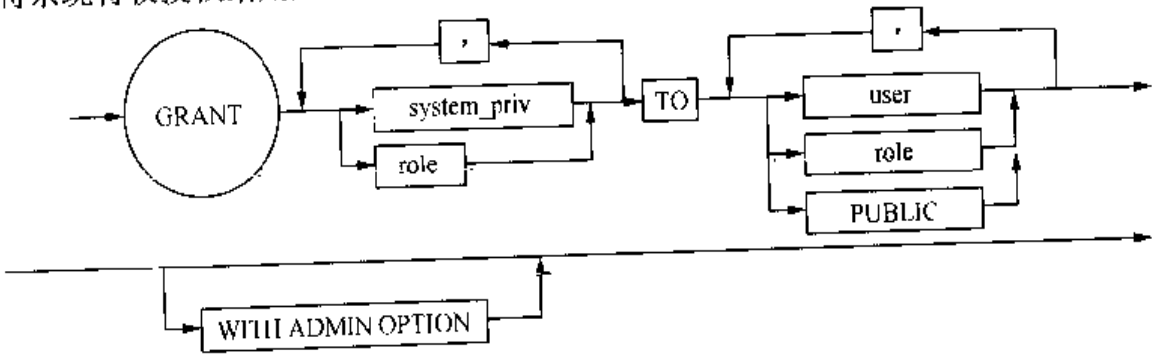
DROP（删除对象或约束）

从数据库中删除对象或约束。使用 DROP 命令需要具有相应的特权。



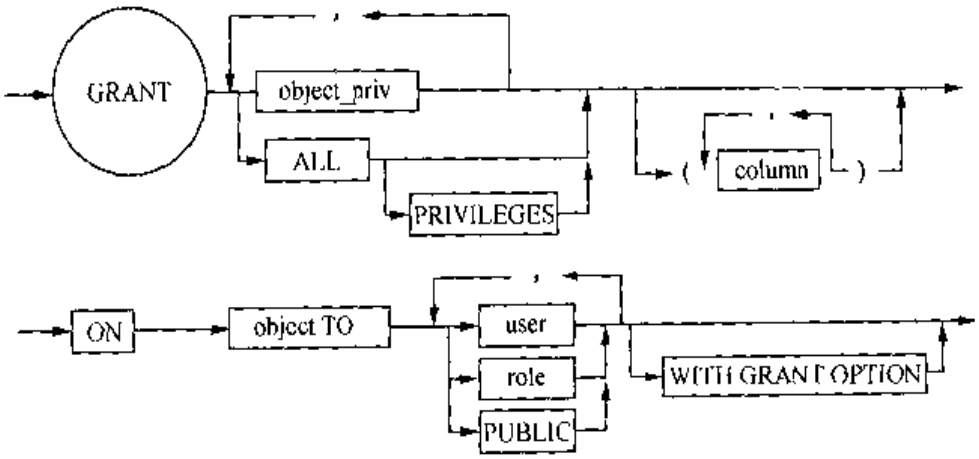
GRANT（系统特权和角色授权）

将系统特权授权给用户和角色。将角色授权给用户和另外的角色。



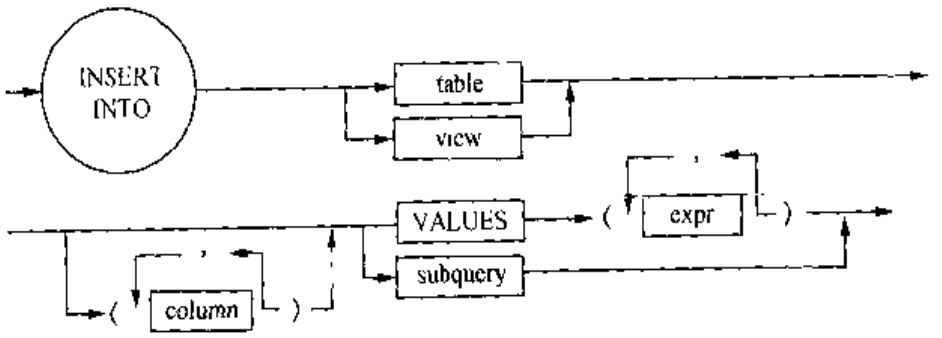
GRANT（对象特权授权）

将某一对象（如表、视图、同义词、包、过程等）的特权授给用户和角色。



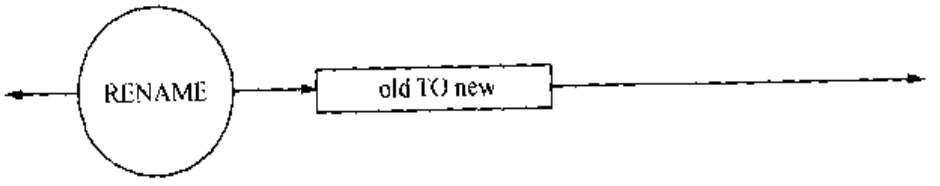
INSERT（插入行）

将若干新行加到表或视图中。



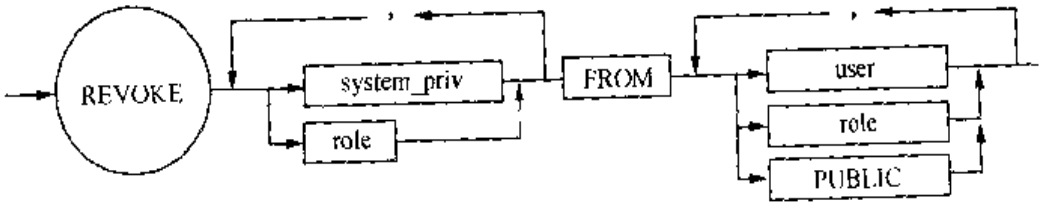
RENAME（重命名）

重新命名表、视图或同义词。



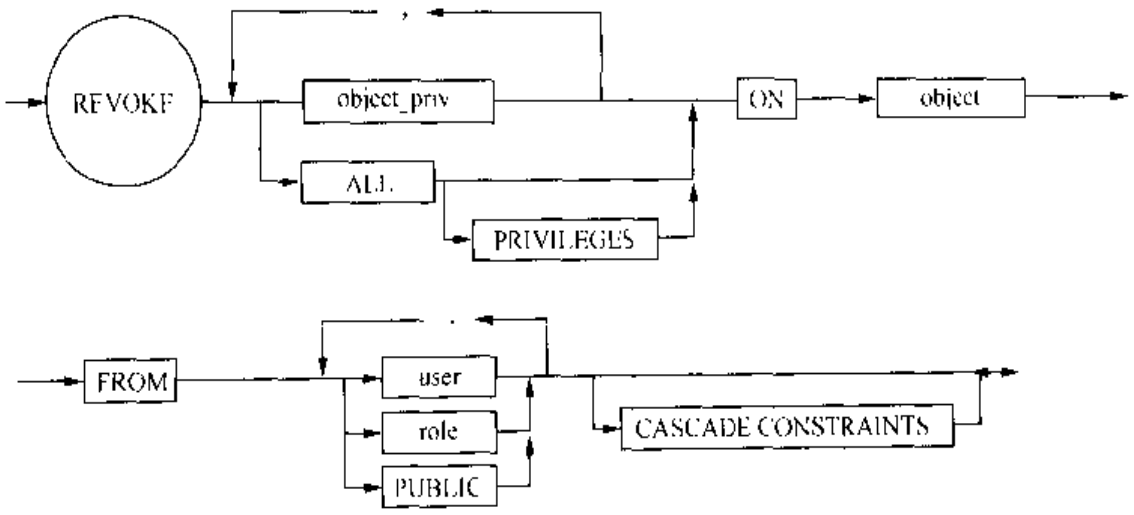
REVOKE（收回系统特权和角色）

从用户和角色收回系统特权和角色。REVOKE 命令与 GRANT 命令功能相反。



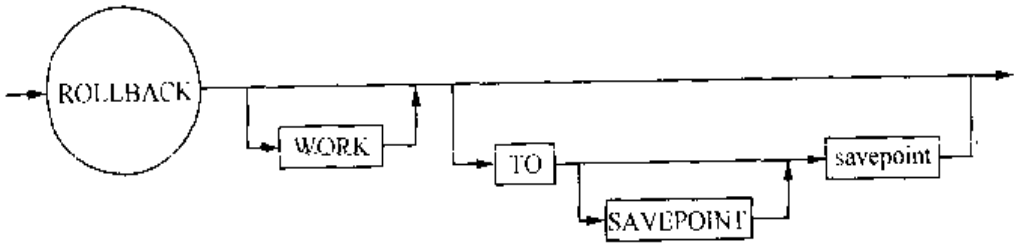
REVOKE（收回对象特权）

从用户和角色收回对象特权和角色。REVOKE 命令与 GRANT 命令功能相反。



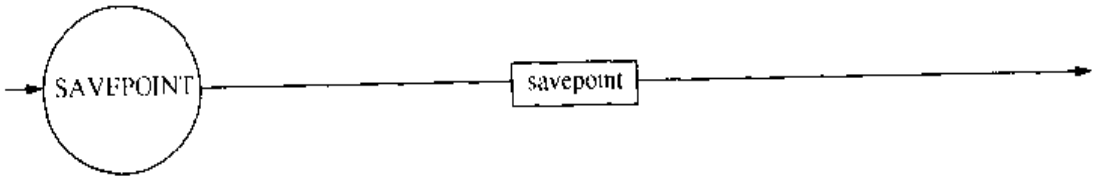
ROLLBACK（事务控制回滚）

撤消该保留点以后对数据库的全部改变。假如未指定保留点，则全部撤消当前事务对数据库的所有改变。



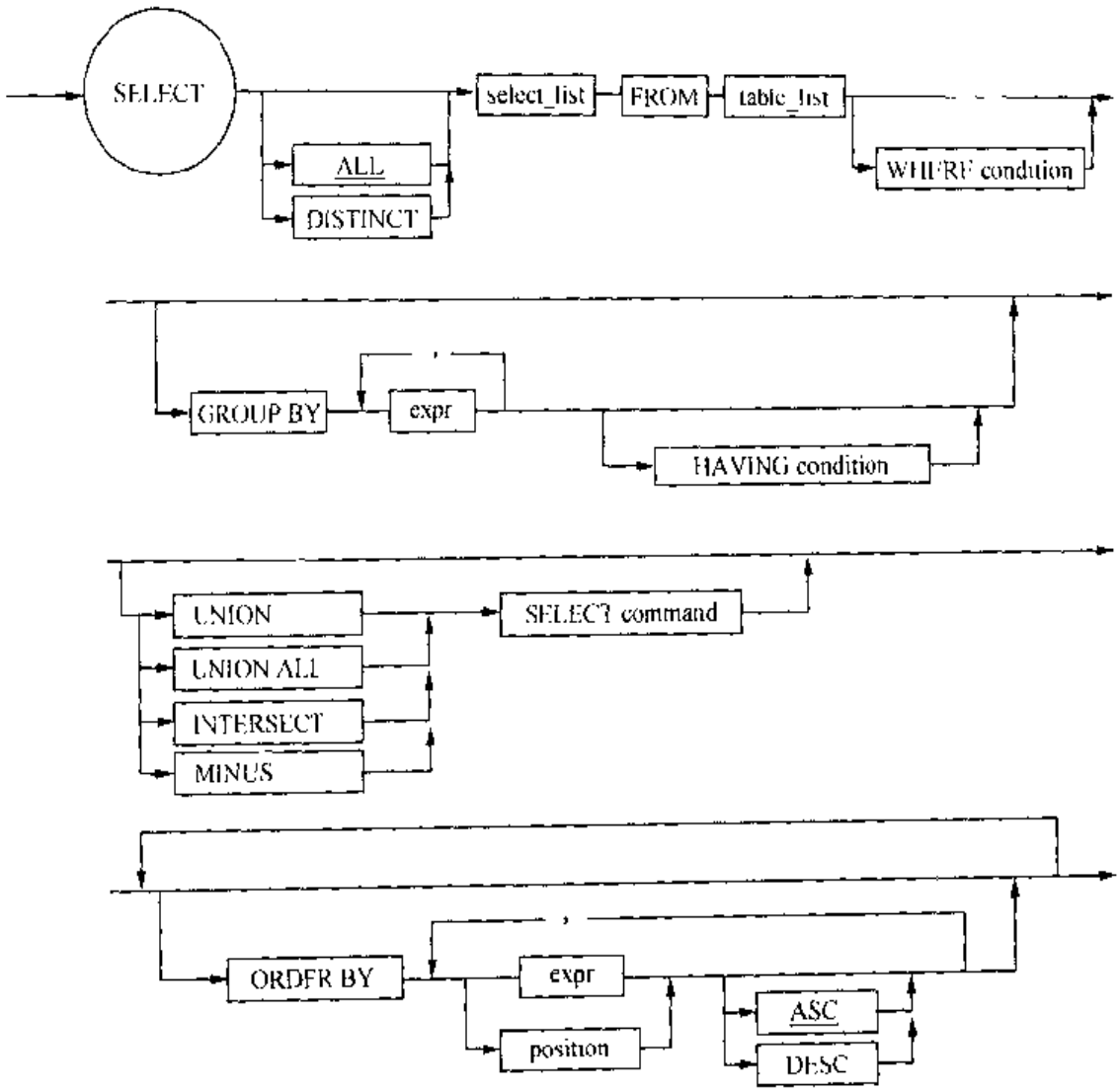
SAVEPOINT（设置事务保留点）

识别当前事务的保留点，以便将来撤消事务对数据库的部分改变。

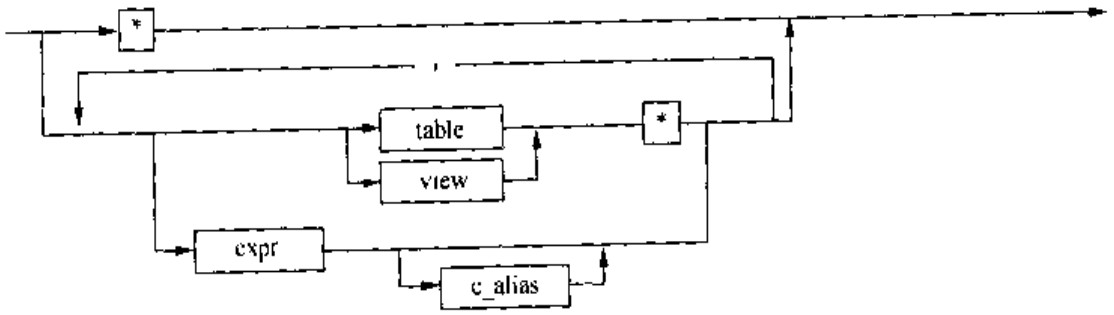


SELECT（查询语句）

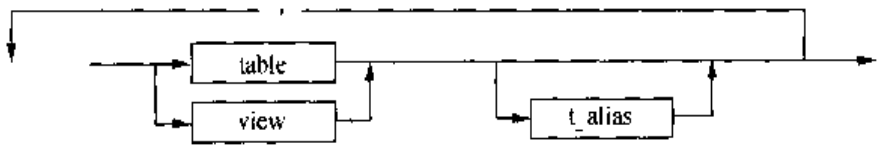
查询一个或多个表或视图，返回某些行和列数据。SELECT可以作为一个语句使用，也可以在另外的语句中作为子查询使用。


SELECT
ALL
DISTINCT
select_list
FROM
table_list
WHERE condition
GROUP BY
expr
HAVING condition
UNION
UNION ALL
INTERSECT
MINUS
SELECT command
ORDER BY
expr
position
ASC
DESC

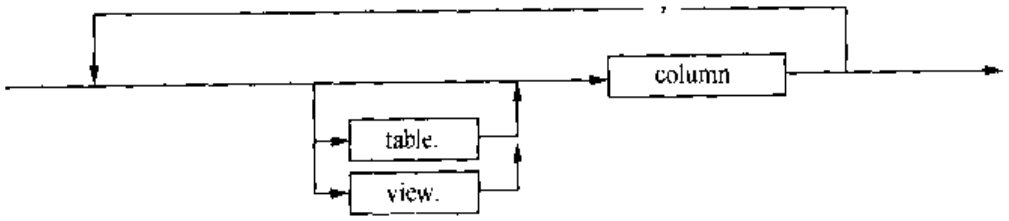
select_list ::=


*
table
view
*
expr
c_alias

table_list ::=



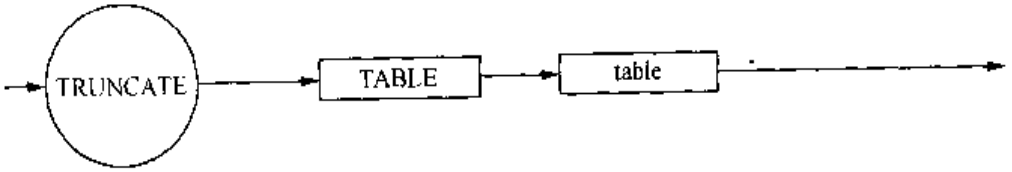
update_list ::=



TRUNCATE

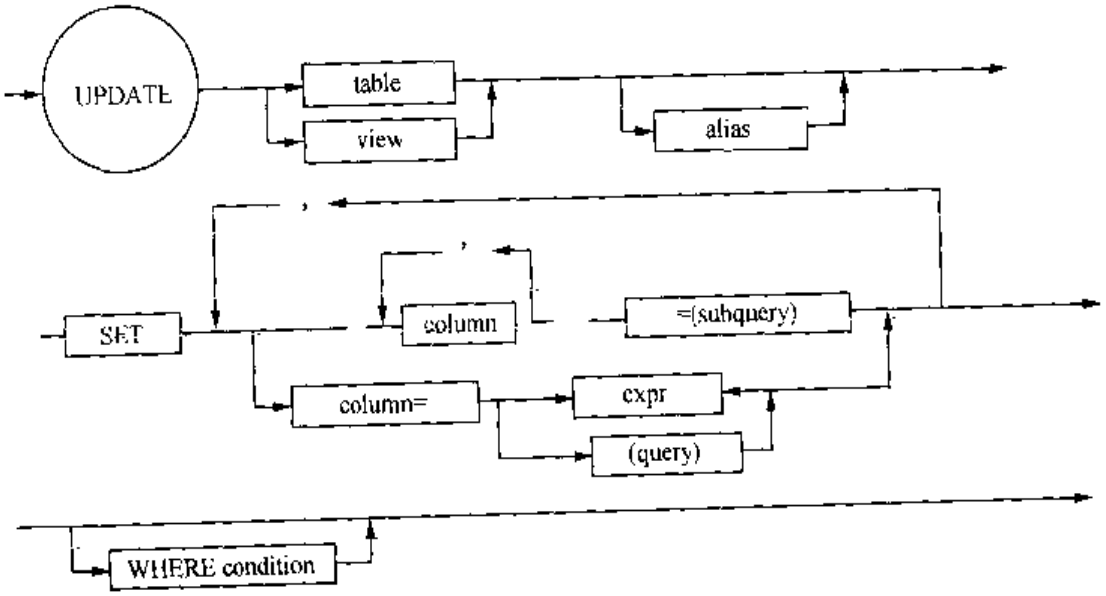
TRUNCATE（删除全部行）

从表或聚集中删除所有的行。



UPDATE（更新数据）

改变满足表或视图中 WHERE 条件的行中的列值。若未指定 WHERE 条件，则修改所有行的列值。



附录 D　E-R 图、运动用品数据库脚本和其他脚本

这是一个原型数据库，运动用品数据库即以它为基础。我们改写了这些数据库表，以适应本书所讨论的范围。可以免费从网址 http://www.cs.jmu.edu/sqldata 下载这些数据库表。E-R 图（参见图 D-1）的复制得到了 Oracle 公司的允许。

```
Rem Sporting Goods Database
Rem
Rem
Rem SCRIPT FUNCTION
Rem Create and populate tables and sequences to support the Sporting Goods business
scenario.
Rem This database is a modified version of the database used in some of the classes of
Rem the ORACLE Corporation.
Rem
Rem NOTES
Rem
Rem MODIFIED
Rem Originally created: GDURHAM Mar 15, 1993--ORACLE Corporation
Rem
Rem Modified and reprinted with permission from The ORACLE Corporation
Rem by Drs.Ramon A.Mata-Toledo and Pauline K.Cushman on Dec 15 ,1999.

set feedback off
Rem if your are using Oracle 7 change the next statement to set compativility v7
set compatibility v8
Rem Start of the creation and population of tables. Please wait.


prompt
prompt creating table s_customer


Rem************** S_CUSTOMER TABLE ********************************
Rem*************************************************************
DROP TABLE s_customer CASCADE CONSTRAINTS;
CREATE TABLE s_customer
(id                             VARCHAR2(3)  CONSTRAINT s_customer_id_nn NOT NULL,
 name                           VARCHAR2(20) CONSTRAINT s_customer_name_nn NOT NULL,
 phone                          VARCHAR2(20) CONSTRAINT s_customer_phone_nn NOT NULL,
 address                        VARCHAR2(20),
 city                           VARCHAR2(20),
 state                          VARCHAR2(15),
 country                        VARCHAR2(20),
 zip_code                       VARCHAR2(15),
 credit_rating                  VARCHAR2(9),
```

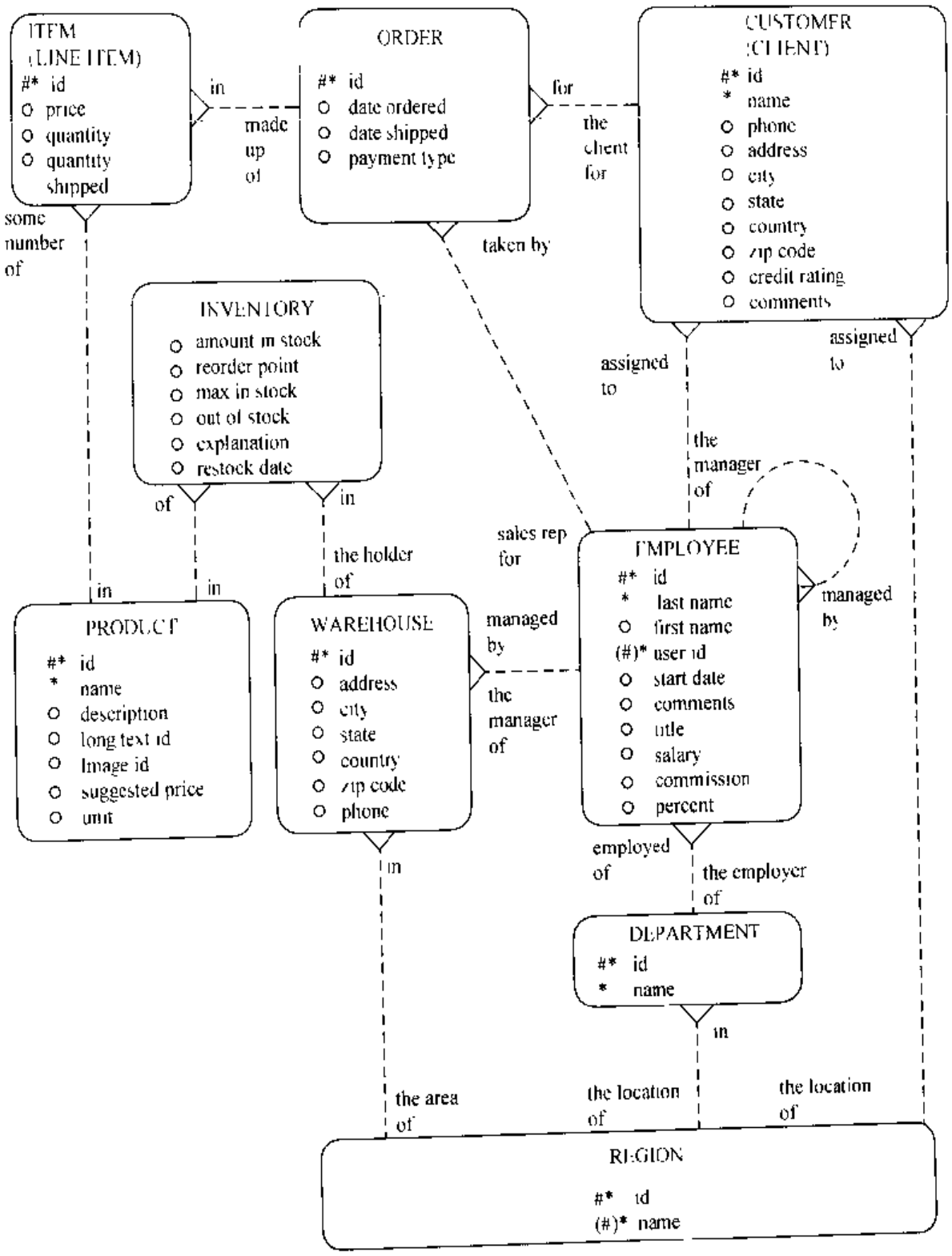

ITEM
(LINE ITEM)
#* id
O price
O quantity
O quantity shipped
in
made up of
ORDER
#* id
O date ordered
O date shipped
O payment type
for
the client for
CUSTOMER
(CLIENT)
#* id
* name
O phone
O address
O city
O state
O country
O zip code
O credit rating
O comments
some number of
taken by
INVENTORY
O amount in stock
O reorder point
O max in stock
O out of stock
O explanation
O restock date
assigned to
assigned to
the manager of
of
in
sales rep for
the holder of
in
in
PRODUCT
#* id
* name
O description
O long text id
O Image id
O suggested price
O unit
WAREHOUSE
#* id
O address
O city
O state
O country
O zip code
O phone
managed by
the manager of
EMPLOYEE
#* id
* last name
O first name
(#)* user id
O start date
O comments
O title
O salary
O commission
O percent
managed by
employed of
the employer of
DEPARTMENT
#* id
* name
in
in
the area of
the location of
the location of
REGION
#* id
(#)* name

图 D-1 运动用品 E-R 图

```
 sales_rep_id                  VARCHAR2(3),
 region_id                     VARCHAR2(3),
 comments                      VARCHAR2(255),
 CONSTRAINT s_customer_id_pk PRIMARY KEY (id),
 CONSTRAINT s_customer_credit_rating_ck
 CHECK (credit_rating IN ('EXCELLENT', 'GOOD', 'POOR'))
);

prompt populating table s_customer

INSERT INTO s_customer VALUES ('301', 'Sports,Inc', '540-123-4567','72 High St',
 Harrisonburg', VA','US', '22809','EXCELLENT', '12', '1', NULL);
INSERT INTO s_customer VALUES ('302', 'Toms Sporting Goods', '540-987-6543','6741
Main St',
'Harrisonburg', 'VA','US', '22809','POOR', '14', '1', NULL);
INSERT INTO s_customer VALUES ('303', 'Athletic Attire', '540-123-6789','54 Market
St',
'Harrisonburg', 'VA','US', '22808','GOOD', '14', '1', NULL);
INSERT INTO s_customer
VALUES ( 304', 'Athletics For All', '540-987-1234','286 Main St', 'Harrisonburg',
'VA', 'US', '22808','EXCELLENT', '12', '1', NULL);
INSERT INTO s_customer VALUES ('305', 'Shoes for Sports', '540-123-9876','538 High
St','Harrisonburg', 'VA','US', '22809','EXCELLENT', '14', '1', NULL);
INSERT INTO s_customer VALUES ('306', 'BJ Athletics', '540-987-9999','632 Water St',
'Harrisonburg', 'VA','US', '22810','POOR', '12 , '1', NULL);

INSERT INTO s_customer VALUES ('403', 'Athletics One', '717-234-6786','912 Columbia
Rd',
'Lancaster', PA','US', 17601','GOOD', '14', '1', NULL);
INSERT INTO s_customer VALUES ('404', 'Great Athletes , '717-987-2341','121 Litiz
Pike',
'Lancaster', 'PA','US', '17602','EXCELLENT', '12', '1', NULL);
INSERT INTO s_customer VALUES ( 405', 'Athletics Two', '717-987-9875','435 High Rd',
'Lancaster', 'PA','US', '17602','EXCELLENT', '14', '1', NULL);
INSERT INTO s_customer VALUES ('406', 'Athletes Attic', '717-234-9888','101
Greenfield Rd',
'Lancaster', 'PA','US', '17601','POOR', '12', '1', NULL);

INSERT INTO s_customer VALUES ('201', 'One Sport', '55-112066222','82 Via Bahia', 'Sao
Paolo',
NULL, 'Brazil , NULL,'EXCELLENT', '12 , '2', NULL);
INSERT INTO s_customer VALUES ('202', 'Deportivo Caracas , '58-28066222','31 Sabana
Grande',
'Caracas', NULL, Venezuela', NULL,'EXCELLENT', '12', '2', NULL);
INSERT INTO s_customer VALUES ('203', 'New Delhi Sports', '91-11903338','11368
Chanakya',
'New Delhi', NULL, 'India', NULL, GOOD', '11', '4', NULL);
INSERT INTO s_customer VALUES ('204', 'Ladysport', '1-206-104-0111','281 Queen
Street',
'Seattle', 'Washington', US', NULL,'EXCELLENT', '11', '1', NULL);
INSERT INTO s_customer VALUES ('205', 'Kim''s Sporting Goods', '852-3693888','15
Henessey Road',
'Hong Kong', NULL, NULL, NULL,'EXCELLENT', '11', '4', NULL);
INSERT INTO s_customer VALUES ('206', 'Sportique', '33-93425722253', 172 Rue de
Place',
```

```
'Cannes', NULL, 'France', NULL,'EXCELLENT', '13', '5', NULL);
INSERT INTO s_customer VALUES ('207', 'Tall Rock Sports', '234-16036222','10 Saint
Antoine',
'Lagos', NULL, 'Nigeria', NULL,'GOOD', NULL, '3', NULL);
INSERT INTO s_customer VALUES ('208', 'Muench Sports', '49-895274449 ','435
Gruenestrasse',
'Munich', NULL, 'Germany', NULL,'GOOD', '13', '5', NULL);

INSERT INTO s_customer VALUES ('209', 'Beisbol Si!', '809-352666','415 Playa Del
Mar',
'San Pedro de Macoris', NULL, 'Dominican Republic', NULL, 'EXCELLENT', '11', '6',
NULL);
INSERT INTO s_customer VALUES ('210', 'Futbol Sonora', '52-404555','5 Via Saguaro',
'Nogales',
NULL, 'Mexico', NULL,'EXCELLENT', '12', '2', NULL);
INSERT INTO s_customer VALUES ('211 , 'Helmut''s Sports', '42-2111222','45 Modrany',
'Prague',
NULL, 'Czechoslovakia', NULL,'EXCELLENT', '11', '5', NULL);
INSERT INTO s_customer VALUES ('212', 'Hamada Sport , '20-31209222','47A Corniche ,
'Alexandria', NULL, 'Egypt', NULL,'EXCELLENT', '13', '3', NULL);
INSERT INTO s_customer VALUES ('213', 'Sports Emporium', '1-415-555-6281','4783
168th Street',
'San Francisco', 'CA', 'US', NULL,'EXCELLENT', '11', '1', NULL);
INSERT INTO s_customer VALUES ('214', 'Sports Retail', '1-716-555-7777','115 Main
Street',
'Buffalo', 'NY', 'US', NULL, 'POOR', '11', '1', NULL);
INSERT INTO s_customer VALUES ('215', 'Sporta Russia', '7-0953892444', 7070
Yekatamina',
'Saint Petersburg', NULL, 'Russia', NULL,'POOR', '11', '5', NULL);
COMMIT;

prompt Finished populating the s_customer table - processing continues
prompt

prompt creating table s_dept
Rem*************** S_CUSTOMER TABLE *********************************
Rem*************************************************************

DROP TABLE s_dept CASCADE CONSTRAINTS;
CREATE TABLE s_dept
(id          VARCHAR2(3)  CONSTRAINT s_dept_id_nn NOT NULL,
 name        VARCHAR2(20) CONSTRAINT s_dept_name_nn NOT NULL,
 region_id  VARCHAR2(3),
 CONSTRAINT s_dept_id_pk PRIMARY KEY (id),
 CONSTRAINT s_dept_name_region_id_uk UNIQUE (name, region_id)
);

prompt populating table s_dept

INSERT INTO s_dept VALUES ('10', 'Finance', '1');
INSERT INTO s_dept VALUES ('31', 'Sales', '1');
INSERT INTO s_dept VALUES ('32', 'Sales', '2');
INSERT INTO s_dept VALUES ('33', 'Sales', '3');
INSERT INTO s_dept VALUES ('34', 'Sales', '4');
INSERT INTO s_dept VALUES ('35', 'Sales', '5');
INSERT INTO s_dept VALUES ('41', 'Operations', '1');
```

```
INSERT INTO s_dept VALUES ('42', 'Operations', '2');
INSERT INTO s_dept VALUES ('43', 'Operations', 3 );
INSERT INTO s_dept VALUES ('44', 'Operations', '4');
INSERT INTO s_dept VALUES ('45', 'Operations', '5');
INSERT INTO s_dept VALUES ('50', 'Administration', '1');
COMMIT;

prompt Finished populating the s_dept table - processing continues
prompt

prompt creating table s_emp
Rem************** S_TEMP TABLE *********************************
Rem*************************************************************

DROP TABLE s_emp CASCADE CONSTRAINTS;
CREATE TABLE s_emp
(id              VARCHAR2(3)  CONSTRAINT s_emp_id_nn NOT NULL,
 last_name       VARCHAR2(20) CONSTRAINT s_emp_last_name_nn NOT NULL,
 first_name      VARCHAR2(20),
 userid          VARCHAR2(8) CONSTRAINT s_emp_userid_nn NOT NULL,
 start_date      DATE CONSTRAINT s_emp_start_date_nn NOT NULL,
 comments        VARCHAR2(255),
 manager_id      VARCHAR2(3),
 title           VARCHAR2(25),
 dept_id         VARCHAR2(3),
 salary          NUMBER(11, 2),
 commission_pct  NUMBER(4, 2),
 CONSTRAINT s_emp_id_pk PRIMARY KEY (id),
 CONSTRAINT s_emp_userid_uk UNIQUE (userid),
 CONSTRAINT s_emp_commission_pct_ck CHECK (commission_pct IN (10, 12.5, 15, 17.5,
 20))
);

prompt populating table s_emp

INSERT INTO s_emp VALUES ('1', 'Martin','Carmen','martincu',TO_DATE('03-MAR-
1990','DD-MON-YYYY'), NULL,NULL, 'President', '50', 4500, NULL);
INSERT INTO s_emp VALUES ('2','Smith','Doris','smithdj',TO_DATE('08-MAR-1990','DD-
MON-YYYY'),NULL,'1', 'VP, Operations','41', 2450,NULL);
INSERT INTO s_emp VALUES ('3', 'Norton','Michael','nortonma',TO_DATE('17-JUN-
1991','DD-MON-YYYY'),NULL,'1', 'VP, Sales', '31', 2400,NULL);
INSERT INTO s_emp VALUES ('4', 'Quentin', 'Mark','quentiml',TO_DATE('07-APR-
1990','DD-MON-YYYY'),NULL,'1', 'VP, Finance', '10', 2450, NULL);
INSERT INTO s_emp VALUES ('5', 'Roper', 'Joseph','roperjm',TO_DATE('04-MAR-
1990','DD-MON-YYYY'),NULL,'1', 'VP, Administration', '50', 2550, NULL);
INSERT INTO s_emp VALUES ('6', 'Brown', 'Molly','brownmr',TO_DATE('18-JAN-
1991','DD-MON-YYYY'),NULL,'2', 'Warehouse Manager', '41', 1600, NULL);
INSERT INTO s_emp VALUES ('7', 'Hawkins', 'Roberta','hawkinrt',TO_DATE( 14-MAY-
1990','DD-MON-YYYY'),NULL,'2', 'Warehouse Manager','42', 1650, NULL);
INSERT INTO s_emp VALUES ('8', 'Burns', 'Ben ','burnsba',TO_DATE('07-APR-1990','DD-
MON-YYYY'),NULL,'2', 'Warehouse Manager','43', 1500, NULL);

INSERT INTO s_emp VALUES ('9', 'Catskill', 'Antoinette','catskiaw',TO_DATE('09-FEB-
1992','DD-MON-YYYY'),NULL,'2', 'Warehouse Manager','44', 1700,NULL);
INSERT INTO s_emp VALUES ('10', 'Jackson', 'Marta','jacksomt',TO_DATE('27-FEB-
1991','DD-MON-YYYY'),NULL,'2', 'Warehouse Manager','45', 1507,NULL);
```

```
INSERT INTO s_emp VALUES ('11', 'Henderson', 'Colin','hendercs',TO_DATE('14-MAY-
1990','DD-MON-YYYY'),NULL,'3', 'Sales Representative', '31', 1400, 10);
INSERT INTO s_emp VALUES ('12', 'Gilson', 'Sam', 'gilsonsj',TO_DATE('18-JAN-
1992','DD-MON-YYYY'),NULL,'3', 'Sales Representative','32', 1490, 12.5);
INSERT INTO s_emp VALUES ('13', 'Sanders', 'Jason', 'sanderjk',TO_DATE('18-FEB-
1991','DD-MON-YYYY'),NULL,'3', 'Sales Representative','33 , 1515, 10);
INSERT INTO s_emp VALUES ('14', 'Dameron', 'Andre , 'dameroap',TO_DATE('09-OCT-
1991','DD-MON-YYYY'),NULL,'3', 'Sales Representative','35', 1450, 17.5);
INSERT INTO s_emp VALUES ('15', 'Hardwick', 'Elaine', 'hardwiem',TO_DATE('07-FEB-
1992','DD-MON-YYYY'),NULL,'6', 'Stock Clerk','41', 1400, NULL);
INSERT INTO s_emp VALUES ('16', 'Brown', 'George', 'browngw',TO_DATE('08-MAR-
1990','DD-MON-YYYY'),NULL,'6', 'Stock Clerk','41', 940, NULL);

INSERT INTO s_emp VALUES ('17', 'Washington , 'Thomas', 'washintl',TO_DATE('09-FEB-
1991','DD-MON-YYYY'),NULL,'7', 'Stock Clerk', '42', 1200, NULL);
INSERT INTO s_emp VALUES ('18', 'Patterson', 'Donald', 'patterdv ,TO_DATE('06-AUG-
1991','DD-MON-YYYY'),NULL,'7', 'Stock Clerk','42', 795, NULL);
INSERT INTO s_emp VALUES ('19', 'Bell', 'Alexander', 'bellag',TO_DATE('26-MAY-
1991','DD-MON-YYYY'),NULL,'8', 'Stock Clerk','43', 850, NULL);
INSERT INTO s_emp VALUES ('20', 'Gantos', 'Eddie', gantosej ,TO_DATE('30-NOV-
1990','DD-MON-YYYY'),NULL,'9', 'Stock Clerk','44', 800, NULL);
INSERT INTO s_emp VALUES ('21', 'Stephenson', 'Blaine', 'stephebs ,TO_DATE('17-MAR-
1991','DD-MON-YYYY'),NULL,'10', 'Stock Clerk','45', 860, NULL);

INSERT INTO s_emp VALUES ('22', 'Chester', 'Eddie', 'chesteek',TO_DATE('30-NOV-
1990','DD-MON-YYYY'), NULL, '9', 'Stock Clerk','44', 800, NULL);
INSERT INTO s_emp VALUES ('23', 'Pearl', 'Roger', 'pearlrg',TO_DATE('17-OCT-
1990','DD-MON-YYYY'), NULL, '9', 'Stock Clerk','34', 795, NULL);
INSERT INTO s_emp VALUES ('24', 'Dancer', 'Bonnie', 'dancerbw',TO_DATE('17-MAR-
1991','DD-MON-YYYY'), NULL, '7', 'Stock Clerk','45', 860, NULL);
INSERT INTO s_emp VALUES ('25', 'Schmitt', 'Sandra', 'schmitss',TO_DATE('09-MAY-
1991','DD-MON-YYYY'), NULL, '8', 'Stock Clerk','45', 1100, NULL);
COMMIT;

prompt Finished populating the s_emp table - processing continues
prompt

prompt creating table s_inventory
Rem*************** S_INVENTORY TABLE *********************************
Rem*************************************************************

DROP TABLE s_inventory CASCADE CONSTRAINTS;
CREATE TABLE s_inventory
(product_id                 VARCHAR2(7)  CONSTRAINT  s_inventory_product_id_nn  NOT
                                         NULL,
 warehouse_id               VARCHAR2(7) CONSTRAINT s_inventory_warehouse_id_nn NOT
                                         NULL,
 amount_in_stock            NUMBER(9),
 reorder_point              NUMBER(9),
 max_in_stock               NUMBER(9),
 out_of_stock_explanation VARCHAR2(255),
 restock_date               DATE,
 CONSTRAINT s_inventory_prodid_warid_pk PRIMARY KEY (product_id, warehouse_id)
 );

 prompt populating table s_inventory
```

```
INSERT INTO s_inventory VALUES ('10011', '101', 650, 625, 1100, NULL, NULL);
INSERT INTO s_inventory VALUES ('10012', '101', 600, 560, 1000, NULL, NULL);
INSERT INTO s_inventory VALUES ('10013', '101', 400, 400, 700, NULL, NULL);
INSERT INTO s_inventory VALUES ('10021', '101', 500, 425, 740, NULL, NULL);
INSERT INTO s_inventory VALUES ('10022', '101', 300, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('10023', '101', 400, 300, 525, NULL, NULL);
INSERT INTO s_inventory VALUES ('20106', '101 , 993, 625, 1000, NULL, NULL);
INSERT INTO s_inventory VALUES ( 20108', '101', 700, 700, 1225, NULL, NULL);
INSERT INTO s_inventory VALUES ('20201', '101', 802, 800, 1400, NULL, NULL);
INSERT INTO s_inventory VALUES ('20510', '101', 1389, 850, 1400, NULL, NULL);
INSERT INTO s_inventory VALUES ('20512', '101', 850, 850, 1450, NULL, NULL);
INSERT INTO s_inventory VALUES ('30321', '101', 2000, 1500, 2500, NULL, NULL);
INSERT INTO s_inventory VALUES ('30326', '101', 2100, 2000, 3500, NULL, NULL);
INSERT INTO s_inventory VALUES ('30421', '101', 1822, 1800, 3150, NULL, NULL);
INSERT INTO s_inventory VALUES ('30426', '101', 2250, 2000, 3500, NULL, NULL);
INSERT INTO s_inventory VALUES ('30433', '101', 650, 600, 1050, NULL, NULL);
INSERT INTO s_inventory VALUES ('32779', '101', 2120, 1250, 2200, NULL, NULL);
INSERT INTO s_inventory VALUES ('32861', '101', 505, 500, 875, NULL, NULL);
INSERT INTO s_inventory VALUES ('40421', '101', 578, 350, 600, NULL, NULL);
INSERT INTO s_inventory VALUES ('40422', '101', 0, 350, 600, 'Phenomenal sales...',
'08-FEB-93');
INSERT INTO s_inventory VALUES ('41010', '101', 250, 250, 437, NULL, NULL);
INSERT INTO s_inventory VALUES ('41020', '101', 471, 450, 750, NULL, NULL);
INSERT INTO s_inventory VALUES ('41050', '101', 501, 450, 750, NULL, NULL);
INSERT INTO s_inventory VALUES ('41080', '101', 400, 400, 700, NULL, NULL);
INSERT INTO s_inventory VALUES ('41100', '101', 350, 350, 600, NULL, NULL);
INSERT INTO s_inventory VALUES ('50169 , '101', 2530, 1500, 2600, NULL, NULL);
INSERT INTO s_inventory VALUES ('50273', '101', 233, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('50417', '101', 518, 500, 875, NULL, NULL);
INSERT INTO s_inventory VALUES ('50418', '101', 244, 100, 275, NULL, NULL);
INSERT INTO s_inventory VALUES ('50419', '101', 230, 120, 310, NULL, NULL);
INSERT INTO s_inventory VALUES ('50530', '101', 669, 400, 700, NULL, NULL);
INSERT INTO s_inventory VALUES ('50532', '101', 0, 100, 175, 'Wait for Spring.',
'12-APR-93');
INSERT INTO s_inventory VALUES ('50536', '101', 173, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('20106', '201', 220, 150, 260, NULL, NULL);
INSERT INTO s_inventory VALUES ('20108', '201', 166, 150, 260, NULL, NULL);
INSERT INTO s_inventory VALUES ('20201', '201', 320, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('20510', '201', 175, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('20512', '201', 162, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('30321', '201', 96, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30326', '201', 147, 120, 210, NULL, NULL);
INSERT INTO s_inventory VALUES ('30421', '201', 102, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30426', '201', 200, 120, 210, NULL, NULL);
INSERT INTO s_inventory VALUES ('30433', '201', 130, 130, 230, NULL, NULL);
INSERT INTO s_inventory VALUES ('32779', '201', 180, 150, 260, NULL, NULL);
INSERT INTO s_inventory VALUES ('32861', '201', 132, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('50169', '201', 225, 220, 385, NULL, NULL);
INSERT INTO s_inventory VALUES ('50273', '201', 75, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50417', '201', 82, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50418', '201 , 98, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50419', '201', 77, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50530', '201', 62, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50532', '201', 67, 60, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('50536', '201', 97, 60, 100, NULL, NULL);
```

```
INSERT INTO s_inventory VALUES ('20510', '301', 69, 40, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('20512', '301', 28, 20, 50, NULL, NULL);
INSERT INTO s_inventory VALUES ('30321', '301', 85, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30421 , '301', 102, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30433', '301 , 35, 20, 35, NULL, NULL);
INSERT INTO s_inventory VALUES ('32779',  301', 102, 95, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('32861', '301', 57, 50, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('40421', '301', 70, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('40422', '301', 65, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41010 , '301', 59, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41020', '301', 61, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41050', '301', 49, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41080', '301', 50, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41100', '301', 42, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('20510', '401', 88, 50, 100, NULL, NULL);
INSERT INTO s_inventory VALUES ('20512', '401', 75, 75, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30321', '401', 102, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30326', '401', 113, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30421', '401', 85, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30426', '401', 135, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('30433', '401', 0, 100, 175, NULL, NULL );
INSERT INTO s_inventory VALUES ('32779', '401', 135, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('32861', '401', 250, 150, 250, NULL, NULL);
INSERT INTO s_inventory VALUES ('40421', '401', 47, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('40422', '401', 50, 40, 70, NULL, NULL);
INSERT INTO s_inventory VALUES ('41010', '401', 80, 70, 220, NULL, NULL);
INSERT INTO s_inventory VALUES ('41020', '401 , 91, 70, 220, NULL, NULL);
INSERT INTO s_inventory VALUES ('41050', '401', 169, 70, 220, NULL, NULL);
INSERT INTO s_inventory VALUES ('41080', '401', 100, 70, 220, NULL, NULL);
INSERT INTO s_inventory VALUES ( 41100', '401', 75, 70, 220, NULL, NULL);
INSERT INTO s_inventory VALUES ('50169', '401', 240, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('50273', '401', 224, 150, 280, NULL, NULL);
INSERT INTO s_inventory VALUES ('50417', '401', 130, 120, 210, NULL, NULL);
INSERT INTO s_inventory VALUES ('50418', '401', 156, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('50419', '401 , 151, 150, 280, NULL, NULL);
INSERT INTO s_inventory VALUES ('50530', '401', 119, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('50532', '401', 233, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('50536', '401', 138, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('10012', '10501', 300, 300, 525, NULL, NULL);
INSERT INTO s_inventory VALUES ('10013', '10501', 314, 300, 525, NULL, NULL);
INSERT INTO s_inventory VALUES ('10022', '10501', 502, 300, 525, NULL, NULL);
INSERT INTO s_inventory VALUES ('10023', '10501', 500, 300, 525, NULL, NULL);
INSERT INTO s_inventory VALUES ('20106', '10501', 150, 100, 175, NULL, NULL);
INSERT INTO s_inventory VALUES ('20108', '10501', 222, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('20201', '10501', 275, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('20510', '10501', 57, 50, 87, NULL, NULL);
INSERT INTO s_inventory VALUES ('20512', '10501', 62, 50, 87, NULL, NULL);
INSERT INTO s_inventory VALUES ('30321', '10501', 194, 150, 275, NULL, NULL);
INSERT INTO s_inventory VALUES ('30326', '10501', 277, 250, 440, NULL, NULL);
INSERT INTO s_inventory VALUES ('30421', '10501', 190, 150, 275, NULL, NULL);
INSERT INTO s_inventory VALUES ('30426', '10501', 423, 250, 450, NULL, NULL);
INSERT INTO s_inventory VALUES ('30433', '10501', 273, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('32779', '10501', 280, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ('32861', '10501', 288, 200, 350, NULL, NULL);
INSERT INTO s_inventory VALUES ( 40421', '10501', 97, 80, 140, NULL, NULL);
```

```
INSERT INTO s_inventory VALUES ('40422', '10501', 90, 80, 140, NULL, NULL);
INSERT INTO s_inventory VALUES ('41010', '10501', 151, 140, 245, NULL, NULL);
INSERT INTO s_inventory VALUES ('41020', '10501', 224, 140, 245, NULL, NULL);
INSERT INTO s_inventory VALUES ('41050', '10501', 157, 140, 245, NULL, NULL);
INSERT INTO s_inventory VALUES ('41080', '10501', 159, 140, 245, NULL, NULL);
INSERT INTO s_inventory VALUES ('41100', '10501', 141, 140, 245, NULL, NULL);
COMMIT;

prompt Finished populating the table s_inventory - processing continues
prompt

prompt creating table s_item
Rem************** S_ITEM TABLE *********************************
Rem*************************************************************

DROP TABLE s_item CASCADE CONSTRAINTS;
CREATE TABLE s_item
(ord_id                 VARCHAR2(3) CONSTRAINT s_item_ord_id_nn NOT NULL,
 item_id                VARCHAR2(7) CONSTRAINT s_item_item_id_nn NOT NULL,
 product_id             VARCHAR2(7) CONSTRAINT s_item_product_id_nn NOT NULL,
 price                  NUMBER(11, 2),
  quantity               NUMBER(9),
  quantity_shipped       NUMBER(9),
  CONSTRAINT s_item_ordid_itemid_pk PRIMARY KEY (ord_id, item_id),
  CONSTRAINT s_item_ordid_prodid_uk UNIQUE (ord_id, product_id)
);

prompt populating table s_item

INSERT INTO s_item VALUES ('100 , '1', '10011', 135, 500, 500);
INSERT INTO s_item VALUES ('100', '2', '10013', 380, 400, 400);
INSERT INTO s_item VALUES ( 100', '3', '10021', 14, 500, 500);
INSERT INTO s_item VALUES ('100', '5', '30326', 582, 600, 600);
INSERT INTO s_item VALUES ('100', '7', '41010', 8, 250, 250);
INSERT INTO s_item VALUES ('100', '6',  30433', 20, 450, 450);
INSERT INTO s_item VALUES ('100', '4',  10023', 36, 400, 400);
INSERT INTO s_item VALUES ('101', '1', '30421', 16, 15, 15);
INSERT INTO s_item VALUES ('101', '3', '41010', 8, 20, 20);
INSERT INTO s_item VALUES ('101 , '5', '50169', 4.29, 40, 40);
INSERT INTO s_item VALUES ('101', '6', '50417', 80, 27, 27);
INSERT INTO s_item VALUES ('101', '7', '50530', 45, 50, 50);
INSERT INTO s_item VALUES ('101', '4', '41100', 45, 35, 35);
INSERT INTO s_item VALUES ('101', '2', '40422', 50, 30, 30);
INSERT INTO s_item VALUES ('102', '1', '20108', 28, 100, 100);
INSERT INTO s_item VALUES ('102', '2', '20201', 123, 45, 45);
INSERT INTO s_item VALUES ('103', '1', '30433', 20, 15, 15);
INSERT INTO s_item VALUES ('103', '2', '32779', 7, 11, 11);
INSERT INTO s_item VALUES ('104', '1', '20510', 9, 7, 7);
INSERT INTO s_item VALUES ('104', '4', '30421', 16, 35, 35);
INSERT INTO s_item VALUES ('104', '2', '20512', 8, 12, 12);
INSERT INTO s_item VALUES ('104', '3', '30321', 1669, 19, 19);
INSERT INTO s_item VALUES ('105', '1', '50273', 22.89, 16, 16);
INSERT INTO s_item VALUES ('105', '3', '50532', 47, 28, 28);
INSERT INTO s_item VALUES ('105', '2', '50419', 80, 13, 13);
INSERT INTO s_item VALUES ( 106', '1', '20108', 28, 46, 46);
INSERT INTO s_item VALUES ('106', '4', '50273', 22.89, 75, 75);
```

```
INSERT INTO s_item VALUES ('106', '5', '50418', 75, 98, 98);
INSERT INTO s_item VALUES ('106', '6', '50419', 80, 27, 27);
INSERT INTO s_item VALUES ('106', '2', '20201', 123, 21, 21);
INSERT INTO s_item VALUES ('106', '3', '50169', 4.29, 125, 125);
INSERT INTO s_item VALUES ('107', '1', '20106', 11, 50, 50);
INSERT INTO s_item VALUES ('107', '3', '20201', 115, 130, 130);
INSERT INTO s_item VALUES ('107', '5', '30421', 16, 55, 55);
INSERT INTO s_item VALUES ('107', '4', '30321 , 1669, 75, 75);
INSERT INTO s_item VALUES ('107', '2', '20108', 28, 22, 22);
INSERT INTO s_item VALUES ('108', '1', '20510', 9, 9, 9);
INSERT INTO s_item VALUES ('108', '6', '41080', 35, 50, 50);
INSERT INTO s_item VALUES ('108', '7', '41100', 45, 42, 42);
INSERT INTO s_item VALUES ('108', '5', '32861', 60, 57, 57);
INSERT INTO s_item VALUES ('108', '2', '20512', 8, 18, 18);
INSERT INTO s_item VALUES ('108', '4', '32779', 7, 60, 60);
INSERT INTO s_item VALUES ( 108', '3', '30321', 1669, 85, 85);
INSERT INTO s_item VALUES ('109', '1', '10011', 140, 150, 150);
INSERT INTO s_item VALUES ('109', '5', '30426', 18.25, 500, 500);
INSERT INTO s_item VALUES ('109', '7', '50418', 75, 43, 43);
INSERT INTO s_item VALUES ('109', '6', '32861', 60, 50, 50);
INSERT INTO s_item VALUES ('109', '4', '30326', 582, 1500, 1500);
INSERT INTO s_item VALUES ('109', '2', '10012', 175, 600, 600);
INSERT INTO s_item VALUES ('109', '3', '10022', 21.95, 300, 300);
INSERT INTO s_item VALUES ('110', '1', '50273', 22.89, 17, 17);
INSERT INTO s_item VALUES ('110', '2', '50536', 50, 23, 23);
INSERT INTO s_item VALUES ('111', '1', '40421', 65, 27, 27);
INSERT INTO s_item VALUES ('111', '2', '41080', 35, 29, 29);
INSERT INTO s_item VALUES ('97', '1', '20106', 9, 1000, 1000);
INSERT INTO s_item VALUES ('97', '2', 30321', 1500, 50, 50);
INSERT INTO s_item VALUES ('98', '1', '40421', 85, 7, 7);
INSERT INTO s_item VALUES ('99', '1', '20510', 9, 18, 18);
INSERT INTO s_item VALUES ('99', '2', '20512', 8, 25, 25);
INSERT INTO s_item VALUES ('99', '3', '50417', 80, 53, 53);
INSERT INTO s_item VALUES ('99', '4', '50530', 45, 69, 69);
INSERT INTO s_item VALUES ('112', '1', '20106', 11, 50, 50);
COMMIT;

prompt Finished populating the s_item table - processing continues
prompt

prompt creating table s_ord
Rem************** S_ORD TABLE *********************************
Rem*************************************************************

DROP TABLE s_ord CASCADE CONSTRAINTS;
CREATE TABLE s_ord
(id                    VARCHAR2(3) CONSTRAINT s_ord_id_nn NOT NULL,
 customer_id           VARCHAR2(3) CONSTRAINT s_ord_customer_id_nn NOT NULL,
 date_ordered          DATE CONSTRAINT s_ord_date_ordered_nn NOT NULL,
 date_shipped          DATE,
 sales_rep_id          VARCHAR2(3),
 total                 NUMBER(11, 2),
 payment_type          VARCHAR2(6) CONSTRAINT s_ord_payment_type_nn NOT NULL,
 order_filled          VARCHAR2(1),
 CONSTRAINT s_ord_id_pk PRIMARY KEY (id),
 CONSTRAINT s_ord_payment_type_ck CHECK (payment_type in ('CASH', 'CREDIT')),
 CONSTRAINT s_ord_order_filled_ck CHECK (order_filled in ('Y', 'N'))
```

```
);

prompt populating table s_ord

INSERT INTO s_ord VALUES ('100 , '204', TO_DATE('31-AUG-1992','DD-MON-YYYY'),
TO_DATE('10-SEP-1992','DD-MON-YYYY'),'11', 601100, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('101', '205', TO_DATE( 31-AUG-1992','DD-MON-YYYY'),
TO_DATE('15-SEP-1992','DD-MON-YYYY'),'14', 8055.6, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('102', '206', TO_DATE( 01-SEP-1992','DD-MON-YYYY'),
TO_DATE('08-SEP-1992','DD-MON-YYYY'), 12', 8335, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('103',  208', TO_DATE( 02-SEP-1992', 'DD-MON-YYYY'),
TO_DATE('22-SEP-1992','DD-MON-YYYY'),'11', 377, 'CASH', 'Y');
INSERT INTO s_ord VALUES ( 104', '208', TO_DATE('03-SEP-1992','DD-MON-YYYY'),
TO_DATE('23-SEP-1992','DD-MON-YYYY'),'13', 32430, 'CREDIT', 'Y');

INSERT INTO s_ord VALUES ('105',  209', TO_DATE('04-SEP-1992','DD-MON-YYYY'),
TO_DATE('18 SEP-1992','DD-MON-YYYY'),'11', 2722.24, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('106', '210', TO_DATE( 07-SEP-1992','DD-MON-YYYY'),
TO_DATE('15-SEP-1992','DD-MON-YYYY'),'12', 15634, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('107',  211', TO_DATE('07-SEP-1992','DD-MON-YYYY'),
TO_DATE('21-SEP-1992','DD-MON-YYYY'),'14', 142171, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('108', '212', TO_DATE('07-SEP-1992','DD-MON-YYYY'),
TO_DATE('10-SEP-1992', DD-MON-YYYY'),'13', 149570, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('109', '213', TO_DATE('08-SEP-1992','DD-MON-YYYY'),
TO_DATE('28-SEP-1992', DD-MON-YYYY'),'11', 1020935, 'CREDIT', 'Y');

INSERT INTO s_ord VALUES ('110', '214', TO_DATE('09-SEP-1992', 'DD-MON-YYYY'),
TO_DATE('21-SEP-1992','DD-MON-YYYY'), 11', 1539.13, 'CASH', 'Y');
INSERT INTO s_ord VALUES ('111', '204', TO_DATE('09-SEP-1992', 'DD-MON-YYYY'),
TO_DATE( 21-SEP-1992','DD-MON-YYYY'),'11', 2770, 'CASH , 'Y');
INSERT INTO s_ord VALUES ('97', '201', TO_DATE('28-AUG-1992','DD-MON-YYYY'),
TO_DATE( 17-SEP-1992','DD-MON-YYYY'),'12', 84000, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('98', '202', TO_DATE('31-AUG-1992','DD-MON-YYYY'),
TO_DATE('10-SEP-1992', DD-MON-YYYY'),'14', 595, 'CASH', 'Y');
INSERT INTO s_ord VALUES ('99', '203', TO_DATE('31-AUG-1992', DD-MON-YYYY'),
TO_DATE('18-SEP-1992','DD-MON-YYYY'),'14', 7707, 'CREDIT', 'Y');
INSERT INTO s_ord VALUES ('112 , '210', TO_DATE('31-AUG-1992','DD-MON-YYYY'),
TO_DATE('10-SEP-1992','DD-MON-YYYY'),'12', 550, 'CREDIT', 'Y');
COMMIT;

prompt Finished populating the s_ord table - Processing continues
prompt

prompt creating table s_product
Rem************** S_PRODUCT TABLE ********************************
Rem*************************************************************

DROP TABLE s_product CASCADE CONSTRAINTS;
CREATE TABLE s_product
(id                          VARCHAR2(7)  CONSTRAINT s_product_id_nn NOT NULL,
 name                        VARCHAR2(25) CONSTRAINT s_product_name_nn NOT NULL,
 short_desc                  VARCHAR2(255),
 suggested_whlsl_price       NUMBER(11, 2),
 whlsl_units                 VARCHAR2(10),
 CONSTRAINT s_product_id_pk PRIMARY KEY (id),
 CONSTRAINT s_product_name_uk UNIQUE (name)
);
```

```
prompt populating table s_product

INSERT INTO s_product VALUES ( 10011', 'Bunny Boot','Beginner''s ski boot',150,
NULL);
INSERT INTO s_product VALUES ('10012', 'Ace Ski Boot','Intermediate ski boot',200,
NULL);
INSERT INTO s_product VALUES ('10013', 'Pro Ski Boot','Advanced ski boot',410,
NULL);
INSERT INTO s_product VALUES ('10021', 'Bunny Ski Pole','Beginner''s ski
pole',16.25, NULL);
INSERT INTO s_product VALUES ('10022', 'Ace Ski Pole','Intermediate ski pole',21.95,
NULL);
INSERT INTO s_product VALUES ( 10023', 'Pro Ski Pole','Advanced ski pole',40.95,
NULL);
INSERT INTO s_product VALUES ('20106', 'Junior Soccer Ball','Junior soccer ball',11,
NULL);
INSERT INTO s_product VALUES ('20108', 'World Cup Soccer Ball','World cup soccer
ball',28, NULL);
INSERT INTO s_product VALUES ('20201', 'World Cup Net','World cup net',123, NULL);
INSERT INTO s_product VALUES ('20510', 'Black Hawk Knee Pads','Knee pads, pair',9,
NULL);
INSERT INTO s_product VALUES ( 20512', 'Black Hawk Elbow Pads','Elbow pads, pair',8,
NULL);
INSERT INTO s_product VALUES ('30321', 'Grand Prix Bicycle','Road bicycle',1669,
NULL);
INSERT INTO s_product VALUES ('30326', 'Himalaya Bicycle','Mountain bicycle',582,
NULL);
INSERT INTO s_product VALUES ('30421', 'Grand Prix Bicycle Tires','Road bicycle
tires',16, NULL);
INSERT INTO s_product VALUES ('30426', 'Himalaya Tires','Mountain bicycle
tires',18.25, NULL);
INSERT INTO s_product VALUES ('30433', 'New Air Pump','Tire pump',20, NULL);
INSERT INTO s_product VALUES ('32779', 'Slaker Water Bottle','Water bottle',7,
NULL);
INSERT INTO s_product VALUES ('32861', 'Safe-T Helmet','Bicycle helmet',60, NULL);
INSERT INTO s_product VALUES ('40421', 'Alexeyer Pro Lifting Bar','Straight bar',65,
NULL);
INSERT INTO s_product VALUES ('40422', 'Pro Curling Bar','Curling bar',50, NULL);
INSERT INTO s_product VALUES ('41010', 'Prostar 10 Pound Weight','Ten pound
weight',8, NULL);
INSERT INTO s_product VALUES ('41020', 'Prostar 20 Pound Weight','Twenty pound
weight',12, NULL);
INSERT INTO s_product VALUES ('41050', 'Prostar 50 Pound Weight','Fifty pound
weight',25, NULL);
INSERT INTO s_product VALUES ('41080', 'Prostar 80 Pound Weight','Eighty pound
weight',35, NULL);
INSERT INTO s_product VALUES ('41100', 'Prostar 100 Pound Weight','One hundred pound
weight',45, NULL);
INSERT INTO s_product VALUES ('50169', 'Major League Baseball','Baseball',4.29,
NULL);
INSERT INTO s_product VALUES ('50273', 'Chapman Helmet','Batting helmet',22.89,
NULL);
INSERT INTO s_product VALUES ('50417', 'Griffey Glove','Outfielder''s glove',80,
NULL);
```

```
INSERT INTO s_product VALUES ('50418', 'Alomar Glove','Infielder''s glove',75,
NULL);
INSERT INTO s_product VALUES ('50419', 'Steinbach Glove', Catcher''s glove',80,
NULL);
INSERT INTO s_product VALUES ('50530', 'Cabrera Bat','Thirty inch bat',45, NULL);
INSERT INTO s_product VALUES ('50532', 'Puckett Bat','Thirty-two inch bat',47,
NULL);
INSERT INTO s_product VALUES ('50536', 'Winfield Bat','Thirty-six inch bat',50,
NULL);
COMMIT;

prompt Finished populating the s_product table - Processing continues
prompt

prompt creating table s_region
Rem************** S_REGION TABLE *********************************
Rem*************************************************************

DROP TABLE s_region CASCADE CONSTRAINTS;
CREATE TABLE s_region
(id           VARCHAR2(3)  CONSTRAINT s_region_id_nn NOT NULL,
 name         VARCHAR2(26) CONSTRAINT s_region_name_nn NOT NULL,
 CONSTRAINT s_region_id_pk PRIMARY KEY (id),
 CONSTRAINT s_region_name_uk UNIQUE (name)
);

prompt populating table s_region

INSERT INTO s_region VALUES ('1', 'North America');
INSERT INTO s_region VALUES ('2', 'South America');
INSERT INTO s_region VALUES ('3', 'Africa / Middle East');
INSERT INTO s_region VALUES ('4', 'Asia');
INSERT INTO s_region VALUES ('5', 'Europe');
INSERT INTO s_region VALUES ('6', 'Central America /Caribbean');
COMMIT;

prompt Finished populating the s_region table - Processing continues
prompt
prompt creating table s_title
Rem************** S_TITLE TABLE *********************************
Rem*************************************************************

DROP TABLE s_title CASCADE CONSTRAINTS;
CREATE TABLE s_title
(title        VARCHAR2(25) CONSTRAINT s_title_title_nn NOT NULL,
 CONSTRAINT s_title_title_pk PRIMARY KEY (title)
);

prompt populating table s_title
INSERT INTO s_title VALUES ('President');
INSERT INTO s_title VALUES ('Sales Representative');
INSERT INTO s_title VALUES ( Stock Clerk');
INSERT INTO s_title VALUES ('VP, Administration');
INSERT INTO s_title VALUES ('VP, Finance');
INSERT INTO s_title VALUES ('VP, Operations');
INSERT INTO s_title VALUES ('VP, Sales');
```

```
INSERT INTO s_title VALUES ('Warehouse Manager');
COMMIT;

prompt Finished populating the s_title table - Processing continues

Rem************** S_WAREHOUSE TABLE *********************************
Rem*************************************************************
prompt
prompt creating table s_warehouse

DROP TABLE s_warehouse CASCADE CONSTRAINTS;
CREATE TABLE s_warehouse
(id             VARCHAR2(7) CONSTRAINT s_warehouse_id_nn NOT NULL,
 region_id      VARCHAR2(3) CONSTRAINT s_warehouse_region_id_nn NOT NULL,
 address        VARCHAR2(20),
 city           VARCHAR2(20),
 state          VARCHAR2(15),
 country        VARCHAR2(20),
 zip_code       VARCHAR2(15),
 phone          VARCHAR2(20),
 manager_id     VARCHAR2(3),
 CONSTRAINT s_warehouse_id_pk PRIMARY KEY (id)
);

prompt populating table s_warehouse
INSERT INTO s_warehouse VALUES ('101', '1', 283 King Street','Seattle', 'WA', 'US',
NULL,
NULL, '6');
INSERT INTO s_warehouse VALUES ('10501', '5','5 Modrany', Bratislava', NULL,
 Slovakia',
NULL, NULL, '10');
INSERT INTO s_warehouse VALUES ('201', '2','68 Via Centrale','Sao Paolo', NULL,
'Brazil',NULL,
NULL, '7');
INSERT INTO s_warehouse VALUES ('301', '3','6921 King Way','Lagos', NULL,
'Nigeria',NULL,
NULL, '8');
INSERT INTO s_warehouse VALUES ('401', '4','86 Chu Street','Hong Kong', NULL,
NULL,NULL,
NULL, '9');
COMMIT;
prompt Finished populating the s_warehouse table - Processing continues

Rem ADD FOREIGN KEYS AND CONSTRAINTS - PLEASE READ NOTE BELOW.

prompt
prompt altering tables to add foreign key constraints
Rem    ************************************************************
Rem    ************************************************************
Rem    THE FOLLOWING CONSTRAINTS SHOULD BE INCLUDED
Rem    AS PART OF THE SPORTING GOODS (SG) SCRIPT. THEY SHOULD
Rem    NOT BE PART OF THE SG_NO_CONSTRAINTS.
Rem    ************************************************************
Rem    ********************** CONSTRAINTS **********************
```

```
ALTER TABLE s_dept
    ADD CONSTRAINT s_dept_region_id_fk
    FOREIGN KEY (region_id) REFERENCES s_region (id);
ALTER TABLE s_emp
    ADD CONSTRAINT s_emp_manager_id_fk
    FOREIGN KEY (manager_id) REFERENCES s_emp (id);
ALTER TABLE s_emp
    ADD CONSTRAINT s_emp_dept_id_fk
    FOREIGN KEY (dept_id) REFERENCES s_dept (id);
ALTER TABLE s_emp
    ADD CONSTRAINT s_emp_title_fk
    FOREIGN KEY (title) REFERENCES s_title (title);
ALTER TABLE s_customer
    ADD CONSTRAINT s_sales_rep_id_fk
    FOREIGN KEY (sales_rep_id) REFERENCES s_emp (id);
ALTER TABLE s_customer
    ADD CONSTRAINT s_customer_region_id_fk
    FOREIGN KEY (region_id) REFERENCES s_region (id);
ALTER TABLE s_ord
    ADD CONSTRAINT s_ord_customer_id_fk
    FOREIGN KEY (customer_id) REFERENCES s_customer (id);
ALTER TABLE s_ord
    ADD CONSTRAINT s_ord_sales_rep_id_fk
    FOREIGN KEY (sales_rep_id) REFERENCES s_emp (id);
ALTER TABLE s_item
    ADD CONSTRAINT s_item_ord_id_fk
    FOREIGN KEY (ord_id) REFERENCES s_ord (id);
ALTER TABLE s_item
    ADD CONSTRAINT s_item_product_id_fk
    FOREIGN KEY (product_id) REFERENCES s_product (id);
ALTER TABLE s_warehouse
    ADD CONSTRAINT s_warehouse_manager_id_fk
    FOREIGN KEY (manager_id) REFERENCES s_emp (id);
ALTER TABLE s_warehouse
    ADD CONSTRAINT s_warehouse_region_id_fk
    FOREIGN KEY (region_id) REFERENCES s_region (id);
ALTER TABLE s_inventory
    ADD CONSTRAINT s_inventory_product_id_fk
    FOREIGN KEY (product_id) REFERENCES s_product (id);
ALTER TABLE s_inventory
    ADD CONSTRAINT s_inventory_warehouse_id_fk
    FOREIGN KEY (warehouse_id) REFERENCES s_warehouse (id);

prompt
prompt Creation and population of database has been completed.
set feedback on
REM MULT.SQL
create table mult
(
        first number,
        secondnumber
);
```

```
insert into mult
values (987,2);
insert into mult
values (22,NULL);
insert into mult
values (NULL,35);
insert into mult
values (5,4);
```

```
REM NAMES.SQL
create table NAMES (
FirstName      varchar2(25),
LastName       varchar2(25)
);
insert into NAMES (FirstName, LastName)
values ('THOMAS', 'JEFFERSON');
insert into NAMES (FirstName, LastName)
values (NULL, 'SOCRATES');
```

```
REM PAY_PERIODS.SQL
create table PAY_PERIODS(
first_check   date,
second_check  date
);
insert into PAY_PERIODS(first_check,second_check)
values ('14-JAN-00','28-JAN-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-FEB-00','29-FEB-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-MAR-00','30-MAR-00');
insert into PAY_PERIODS(first_check,second_check)
values ( 14-APR-00','28-APR-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-MAY-00','30-MAY-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-JUN-00','30-JUN-00');
insert into PAY_PERIODS(first_check,second_check)
values ('14-JUL-00','28-JUL-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-AUG-00','30-AUG-00');
insert into PAY_PERIODS(first_check,second_check)
values ('15-SEP-00','29-SEP-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-OCT-00','30-OCT-00');
insert into PAY_PERIODS(first_check,second_check)
values ('16-NOV-00','30-NOV-00');
insert into PAY_PERIODS(first_check,second_check)
values ('15-DEC-00','29-DEC-00');
```

```
REM PROGRAMMER.SQL
Create table Programmer
(EmpNo Varchar2(3)                    PRIMARY KEY,
 Last_Name Varchar2(25)     NOT NULL,
 First_Name Varchar2(25),
 Hire_Date Date,
 Project Varchar2(3),
 Language Varchar2(15),
 TaskNo Number(2),
 Clearance Varchar2(25)
);
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('201','Campbell', 'John', '1-JAN-95',
       'NPR', 'VB',52, 'Secret');
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('390', 'Bell', 'Randall', '1-MAY-93',
       'KCW', 'Java', 11, 'Top Secret');
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('789', 'Hixon', 'Richard', '31-AUG-98 ,
       'RNC', 'VB', 11, 'Secret');
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('134','McGurn', 'Robert', '15-JUL-95',
       'TIP','C++',52, 'Secret');
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('896','Sweet', 'Jan', '15-JUN-97',
       'KCW','Java',10, 'Top Secret');
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('345',' Rowlett', 'Sid', '15-NOV-99',
       'TIP','Java',52, NULL);
INSERT INTO programmer (EmpNo, Last_Name, First_Name, Hire_Date,
       Project, Language, TaskNo, Clearance)
Values('563', 'Reardon', 'Andy', 15-AUG-94',
        NIT', 'C++', 89, 'Confidential');
```

```
REM STATS.SQL
CREATE TABLE stats
( even        number,
  odd         number
);
insert into stats values(2,1);
insert into stats values(NULL,3);
insert into stats values(6,5);
insert into stats values(8, NULL);
insert into stats values(10,9);
```

```
insert into stats values(NULL,11);
insert into stats values(14,NULL);
insert into stats values(16,15);
```

```
REM TRYNUM.SQL
create table trynum
(
        first number,
        second number,
        third number
);

insert into trynum
values (987,987,987);
insert into trynum
values (987.222,987,987.23);
insert into trynum
values (98.5,99,98.5);
insert into trynum
values (98765,98765,98765);
insert into trynum
values (23.987,24,23.99);
insert into trynum
values (.00003,0,0);
insert into trynum
values (100.9,101,100.9);
insert into trynum
values (.00005,0,0);
insert into trynum
values (1.9,2,1.9);
insert into trynum
values (10.1,10,10);
```

```
REM WORLD_CITIES.SQL
create table World_Cities (
City          varchar2(25),
Country       varchar2(25),
Continent     varchar2(25),
Latitude      number,
NorthSouth    char(1),
Longitude     number,
EastWest      char(1)
);
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('ATHENS','GREECE','EUROPE',37.59,'N',23.44,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
```

```
values ('ATLANTA ','UNITED STATES','NORTH AMERICA' ,
       33.45,'N',84.23,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('DALLAS ','UNITED STATES','NORTH AMERICA',
       32.47,'N',96.47, W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('NASHVILLE','UNITED STATES','NORTH AMERICA',
       36.09,'N',86.46,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('VICTORIA','CANADA','NORTH AMERICA',
       48.25,'N',123.21,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('PETERBOROUGH','CANADA','NORTH AMERICA',
       44.18,'N',79.18,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('VANCOUVER','CANADA','NORTH AMERICA',
       49.18,'N',123.04,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('TOLEDO','UNITED STATES','NORTH AMERICA',
       41.39,'N',83.82,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('WARSAW','POLAND','EUROPE',52.15,'N',21.00,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('LIMA','PERU','SOUTH AMERICA',12.03,'S',77.03,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('RIO DE JANEIRO','BRAZIL','SOUTH AMERICA',
       22.43,'S',43.13,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('SANTIAGO','CHILE ','SOUTH AMERICA',33.27,'S',70.40,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('BOGOTA','COLOMBIA','SOUTH AMERICA',
       04.36,'N',74.05,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('BUENOS AIRES','ARGENTINA','SOUTH AMERICA',
       34.36,'S',58.28,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                          NorthSouth,Longitude,EastWest)
values ('QUITO','ECUADOR','SOUTH AMERICA',00.13,'S',78.30,'W');
insert into World_Cities (City,Country,Continent,Latitude,
```

```
                        NorthSouth,Longitude,EastWest)
values ('CARACAS','VENEZUELA','SOUTH AMERICA',
       10.30,'N',66.56,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('MADRAS','INDIA','ASIA',13.05,'N',80.17,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('NEW DEHLI','INDIA','ASIA',28.36,'N',77.12,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('BOMBAY','INDIA','ASIA',18.58,'N',72.50,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('MANCHESTER','ENGLAND','EUROPE',53.30,'N',2.15,'W');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('LONDON','ENGLAND','EUROPE',51.30,'N',0.0,NULL);
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('MOSCOW','RUSSIA','EUROPE',55.45,'N',37.35,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('PARIS','FRANCE','EUROPE',48.52,'N',2.20,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('SHENYANG','CHINA','ASIA',41.48,'N',123.27,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('CAIRO','EGYPT','AFRICA',30.03,'N',31.15,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('TRIPOLI','LYBIA','AFRICA',32.54,'N',13.11,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('BEIJING','CHINA','ASIA',39.56,'N',116.24, E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('ROME','ITALY','EUROPE',41.54,'N',12.29,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ( TOKYO','JAPAN','ASIA',35.42,'N',139.46,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('SYDNEY','AUSTRALIA','AUSTRALIA',33.52,'S',151.13,'E');
insert into World_Cities (City,Country,Continent,Latitude,
                        NorthSouth,Longitude,EastWest)
values ('SPARTA','GREECE','EUROPE',37.05,'N',22.27,'E');
insert into World_Cities (City,Country,Continent,Latituden,
                        NorthSouth,Longitude,EastWest)
values ('MADRID','SPAIN','EUROPE',40.24,'N',3.41,'W');
```

附录 E　用 SQL * Plus 命令创建报表

在第 2 章中，我们学习了一些系统变量以及在显示查询结果时改变表头的 COLUMN 命令的使用。通过使用 SQL * plus 的附加功能，可以产生将表头和数据列格式化的报表。在这个附录中，我们将把精力集中放在使用 SQL * Plus 产生报表上。

系统变量和 SET 命令

正如在附录 A 所看到的，SET 命令可用来为系统变量或环境变量（sysvars）赋值。表 E-1列出了对产生报表或查询最有用的一些系统变量。

表 E-1　对格式化报表有用的部分系统变量列表

系统变量	用途及缺省值
LINESIZE	每行的最大字符数，缺省值是 80
PAGESIZE	在一页中能显示的最大行数，缺省值是 24
NEWPAGE	在一页顶行之前的行数，缺省值是 1
NUMWIDTH	数值类型值的缺省列宽度
SPACE	在报表中列之间的空格数。允许最大值是 10

为系统变量赋值的语法如下：

```
SET system-variable new-value
```

格式化命令

除了系统变量外，一些 SQL * Plus 命令对创建报表也很有帮助。表 E-2 列出了这些命令的一部分。所有这些命令必须先于定义报表格式的查询。

表 E-2　部分有用的 SQL * Plus 命令

命令名称	描述
BTITLE [title|system var]	在页的底部放置标题或系统变量
TTITLE [title|system var]	在页的顶部放置标题或系统变量
BREAK	在报表中指定格式在何处和如何变化，下面进一步解释
COMPUTE	计算和打印若干行子集的摘要，下面进一步解释

下面的例子说明了这些命令的用法。

例 1

在下面显示查询的结果。确信在这页的顶部读到标题“Sales Representatives of the SG Company”。

```
SELECT first_name, last_name, dept_id AS "Dpt"
FROM s_emp ←——————— 在此例中，使用该查询语句。
WHERE title = 'Sales Representative';
```

为产生所需要的表标题，要使用 TTITLE 命令，如下所示。注意标题是用单引号括起来的。如前所述，TTITLE 命令必须先于查询的执行。查询的结果和 TTITLE 命令的效果如下。结果被格式化了，以适应这页的宽度。

```
SELECT first_name, last_name, dept_id AS "Dpt"   TTITLE的效果
FROM s_emp
WHERE title = 'Sales Representative';
Wed Feb 09                                              page 1
                Sales Representatives of the SG Company

FIRST_NAME           LAST_NAME            Dpt
-------------------- -------------------- ---
Colin                Henderson            31
Sam                  Gilson               32
Jason                Sanders              33
Andre                Dameron              35
```

通过观察可以发现，除显示了给出的标题 TTITLE 命令外，作为缺省，在页的左上角和右上角还有系统日期和页号。

BREAK 命令的使用[①]

BREAK 命令用于控制报表的外观，因为它允许我们规定在某一列上控制截断报表。当

① 运行本节和下一节的查询之前，读者应刷新 SG 数据库。

控制列的值发生变化时，控制截断发生。也就是说，对于被选作控制变量的值来说，当一行的值与前一行相应的值不同时就会发生控制截断。

命令的基本语法如下：

```
BREAK ON report-element [action [action]. . . . . ]
```

Where report-element requires the following syntax:

```
{column | expression | REPORT}
```

Where Action requires the following syntax:

```
[SKIP n | SKIP PAGE ] [NO DUPLICATES | DUPLICATES]
```

一旦发出 BREAK 命令，它就一直有效，直到有另一个 BREAK 命令对其进行重定义，或者用 CLEAR BREAKS 命令使其失效为止。无论何时用 BREAK 命令创建报表，用户都应该把如下的准则记在心里：

- 当与 SELECT 语句的 ORDER BY 子句一起使用时，BREAK 命令能够发挥最佳作用。
- 查询必须按 BREAK 命令指定的列（ORDER BY）排序。
- 在任何时间只能有一个 BREAK 命令有效。然而，单一的命令能设置多个控制截断。

下面的例子说明 BREAK 命令的用法，该命令与一个查询连接，查询用 BREAK 指定的列排序

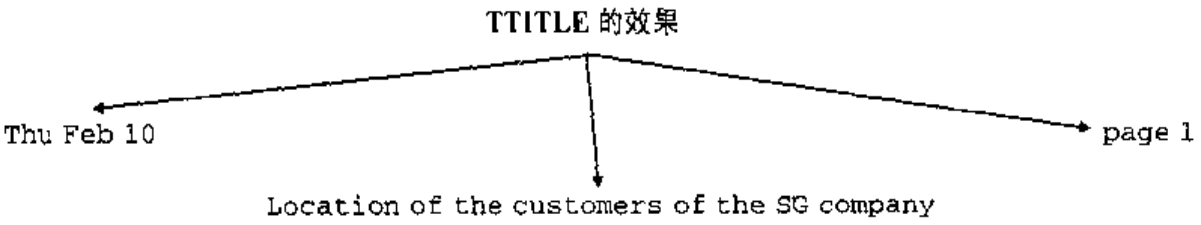
例 2

显示运动用品数据库中的所有顾客姓名、所在城市、国家和地区，将顾客按地区分组。该查询及其结果如下。为了说明及适合一页的宽度，该查询结果被格式化了。

```
TTITLE ' Location of the customers of the SG Company'

BTITLE 'Internal Information'

BREAK on country SKIP 2

SELECT region_id, country, city, name

FROM s_customer

ORDER BY region_id, country;
```

注意 BREAK 命令的使用和按 break 列排序的 ORDER BY 子句

注意列的顺序（从左到右），并和报表相比较



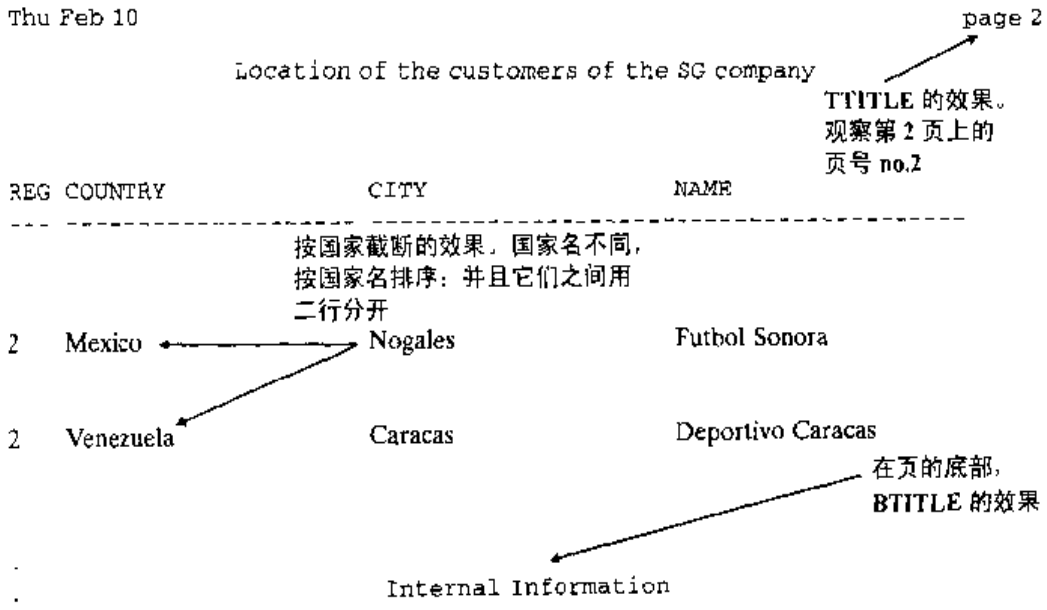
REG	COUNTRY	CITY	NAME
1	US	Harrisonburg	Sports, Inc
1		Harrisonburg	Toms Sporting Goods
1		Harrisonburg	Athletic Attire
1		Harrisonburg	Athletics For All
1		Harrisonburg	Shoes for Sports
1		Harrisonburg	BJ Athletics
1		Lancaster	Athletics One
1		Seattle	Ladysport
1		Lancaster	Great Athletes
1		Buffalo	Sports Retail
1		San Francisco	Sports Emporium
1		Lancaster	Athletics Two
1		Lancaster	Athletes Attic
2	Brazil	Sao Paolo	One Sport

选项 SKIP2 的效果，在一个截断值后空 2 行

首先按地区排序，再在同一地区内按国家排序的效果

注意，在国家名变化时截断

Internal Information ← BTITLE 的效果



让我们来检查这些命令和与每条命令各部分紧密联系的效果。为达到说明的目的，我们复制了这些命令。

```
TTITLE ' Location of the customers of the SG company'
BTITLE 'Internal Information'
BREAK on country SKIP 2
SELECT region_id, country, city, name
FROM s_customer
ORDER BY region_id, country;
```

每个命令各自的效果说明如下：

- TTITLE 命令指示系统在每页的开始或顶部打印由单引号括起来的表标题。该命令还会在页的左上角打印出系统日期，并在右上角打印页号。
- BTITLE 命令指示系统在每页的底部或末端打印由单引号括起来的信息，条件是打印了 PAGESIZE 行数。
- BREAK 命令通过 ON 子句，指示 country 列是截断列。SKIP 2 选项指示系统在每个截断后打印（跳过）2 个空行。也就是说，无论什么时候，在截断列的值有变化时就打印。
- SELECT 语句指示系统从 S_EMP 表检索 SELECT 子句中所述的各数据列的值。ORDER BY 子句用来设定截断列（country）的行顺序。

为了使报表的可读性更强，总是将列选择为从左到右、从高层到底层。例如，在这种情况下，我们选择地区作为顶层，因为地区包括国家；然后选择国家，因为国家包括城市；然后，选择城市。我们还可以继续选择什么，像各城市内的市区，在市区内我们可以选择街区，在街区内还可以选择街道等等。

在报表中的计算

COMPUTE命令能用来实现在报表中的计算。无论什么时候，控制截断一出现，就实现一次计算。只有与BREAK命令联系在一起，COMPUTE命令才能工作，否则就失效。因此，COMPUTE命令必须跟随一个BREAK命令。通常使用一个函数来实现COMPUTE命令的计算（见表E-3）。

COMPUTE命令的基本格式如下：

COMPUTE [function] OF { column | alias} ON {break-column | REPORT} …]

在COMPUTE命令中的部分最常用函数如表E-3所示。

表E-3 COMPUTE命令中使用的部分常用函数列表

函数名	用途	函数应用的数据类型
AVG	非空值的平均值	数值型
COUNT	非空值个数计数	所有的数据类型
MAXIMUM	最大值	数值型/字符型
MINIMUM	最小值	数值型/字符型
SUM	非空值的和	数值型

用户总能在SQL提示符后发出如下命令，来核实是否有一个有效的计算命令，或者去除任何一个有效的计算命令：

COMPUTE

或

CLEAR COMPUTES

例3

在S_EMP表中，显示所有雇员的工作职务、姓氏和工资，计算每组相同职务的工资子合计。假定图E-1的环境设置是有效的。

```
TTITLE 'Employee Salaries by Title'
BTITLE 'For Internal Use Only'
COLUMN salary HEADING 'Salary' FORMAT $99,999.99
```

图E-1 环境设置

产生需要报表的相应BREAK和COMPUTE命令、SQL查询以及查询的部分结果如下所示。观察BREAK和COMPUTE命令组合的效果。这个结果被格式化了，以适应页的宽度和便于解释。

```
CLEAR COMPUTES
BREAK ON title SKIP 2
COMPUTE SUM OF salary on TITLE
SELECT title, last_name, salary
FROM s_emp
ORDER BY title, last_name, salary
```

```
Thu Feb 10                                                    page 1

                        Employee Salaries by Title

                                                    注意按职务的工资合计
TITLE                     LAST_NAME                 Salary
------------------------- ------------------------- ----------
President                 Martin                    $4,500.00
**************************                          ----------
sum                                                 $4,500.00
                          观察在职务列的截断

Sales Representative      Dameron                   $1,450.00
                          Gilson                    $1,490.00
                          Henderson                 $1,400.00
                          Sanders                   $1,515.00
**************************                          ----------
sum                                                 $5,855.00
Stock Clerk               Bell                        $850.00
                          Brown                       $940.00
                          Chester                     $800.00
                          Dancer                      $860.00
                                                    BTITLE 的效果
.
.                         For Internal Use Only
.
.
25 rows selected.
```

为清除任何一个有效的 COMPUTE 命令，需要用 CLEAR COMPUTES 命令。不需要 CLEAR BREAKS 命令，这是因为在任一给定时间，仅有一个 BREAK 命令是有效的，这在前面已经讲过。因此，这一新的 BREAK 命令会重新定义任何一个前面有效的 BREAK 命令。

为说明起见，我们对下面 COMPUTE 命令中所包含的各语法成分进行讨论：

```
COMPUTE SUM OF salary ON TITLE
```

在该命令中可以看到：

- SUM 函数指示打印实施计算的结果。
- OF salary 子句指明求和计算的列名。
- ON TITLE 子句指明控制截断出现的列名。

简要地说，职务每发生变化（截断）时，该命令都计算有相同职务雇员的工资之和。注意，在该报表中没有工资的总计。总计的含义是各组工资的总和。总计放在报表的结尾处。例如，若报表中有总计，总计应该是全部工资之和。如果需要在报表中得到这个总和，用户必须在 BREAK 和 COMPUTE 命令中用 REPORT 选项，下面讲解了说明。这里仅展示了两条命令，因为我们假定其余的语句与先前的查询是相同的。

```
BREAK ON title SKIP 2 ON REPORT
COMPUTE SUM OF salary ON TITLE REPORT
```

注意在 BREAK 和 COMPUTE 命令中 REPORT 子句的使用

为说明 REPORT 选项在 BREAK 和 COMPUTE 两命令中的用法，报告最后一页的部分，输出如下。查询结果被格式化了，以适应页的宽度。

```
Thu Feb 10                                                    page 4

                        Employee Salaries by Title

TITLE                          LAST_NAME                        Salary
------------------------------ ------------------------------ ----------
Warehouse Manager              Hawkins                       $1,650.00
                               Jackson                       $1,507.00
                                                             ----------
**************************                                   $7,957.00
sum

                                                             ----------
                                                             $38,562.00
sum
```

在 BREAK 和 COMPUTE 两命令中使用 REPORT 选项产生的总计

所有仓库经理的工资子合计，注意这个经理列表是不完整的

TTITLE 和 BTITLE 命令的其他特性

到目前为止，无论什么时候，我们所用的 TTITLE 和 BTITLE 命令都是系统的缺省值。

表 E-4 列出了标题命令所带的部分选项，以对标题外观发挥更多的控制。

表 E-4　TTITLE 和 BTITLE 命令部分选项列表

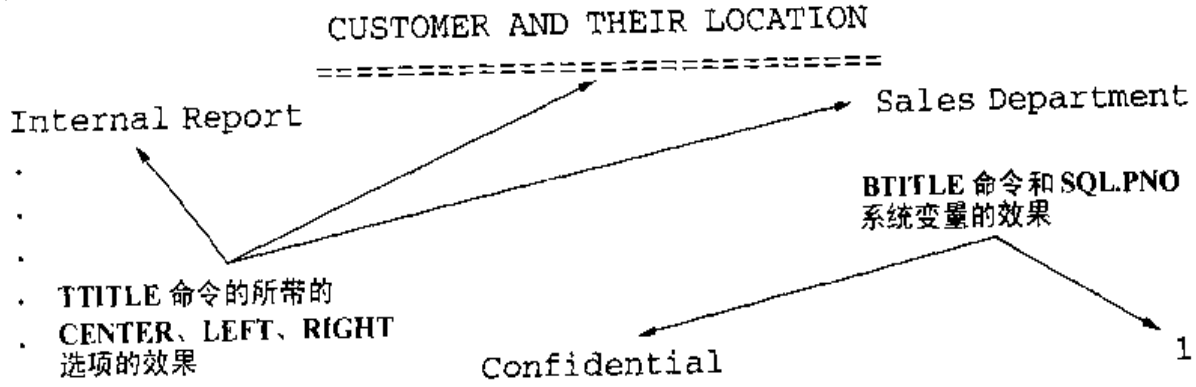
选项	说明
COL n	到 n 列
SKIP n	在打印下一个字符前，空 n 行
LEFT	后续字符序列左对齐
CENTER	字符序列居中
RIGHT	后续字符序列右对齐

有时，标题可能会超过 80 个字符，需要在下一行继续写。为此，要用连字符，让系统知道当前行要继续到下一行。

标题命令与表 E-4 中任一选项一起使用时的一个副作用是，系统不再在页的顶部或底部打印系统日期和页号。为了显示页号，要将页号系统变量 SQL.PNO 作为 TTITLE 命令的一部分。附录的后面将继续说明如何在报表中包括系统日期。下面的命令说明了某些系统变量和表 E-4 中一些选项的用法。

```
TTITLE CENTER 'CUSTOMER AND THEIR LOCATION' SKIP 1 -
CENTER '============================' SKIP 1 -
LEFT 'Internal Report' RIGHT 'Sales Department' SKIP 2
BTITLE CENTER 'Confidential' RIGHT SQL.PNO
```

假定仍使用本附录例 2 的查询，且其他的设置都有效。报表的第 1 页顶部和底部的部分输出如下所示。注意，为适应页的宽度，报表被格式化了。



可以看到，仅一个 TTITLE 命令就扩展了 3 行。在这种情况下我们用了连字符，指示该命令在下一行继续。CENTER 选项跟着 28 个“=”字符，从而使报表的主标题得到了下划线效果。LEFT 和 RIGHT 选项的结合，产生了标题的第二行。SKIP 选项允许标题打印在一行以上。使用系统变量 SQL.PNO 作为 BTITLE 命令的一部分，允许打印报表的页号。

用 BTITLE 命令的 TTITLE 打印系统日期比使用系统变量 SQL.PNO 稍复杂一些。要打印系统日期，需要同时使用 NEW_VALUE 命令、COLUMN 命令的几个新选项以及表别名。考虑下面的命令序列。注意，只列出了那些受影响的命令，其余的命令和选项仍然有

效。

```
COLUMN today_date NEW_VALUE today NOPRINT FORMAT A1 TRUNC
BTITLE CENTER 'Confidential' RIGHT today
.
.
.
SELECT title, last_name, salary,TO_CHAR(SysDate) "Today_date"
FROM s_emp
ORDER BY title, last_name, salary;
```

首先，可以看到 SysDate 系统变量作为 SELECT 语句的伪列被打印出来。在打印之前，先将它转换成字符变量，给伪列一个别名 today _ date。还应注意到这个别名在 COLUMN 命令中的使用。SQL * Plus 的 NEW _ VALUE 命令用来给用户变量 today 赋值 today _ date。这个值用在后面的 BTITLE 命令中。NOPRINT 选项指示系统当打印查询结果时，不打印 today _ date 的列值。最后，应注意到在格式化子句 FORMAT A1 TRUNC 中 TRUNC 的应用。这个选项以及 FORMAT A1 的使用需要做些说明。读者应该知道，当日期用 TO _ CHAR 函数格式化时，系统用缺省的列宽 100 个字符。因此，当确定 LINESIZE 是否超过列宽时，有必要欺骗一下系统，以使它不将 today 列的宽度考虑在内。选项 FORMAT A1 TRUNC 欺骗系统，使用像 TO _ CHAR 格式所指定的同样多的字符来替代系统的缺省值。

在 BTITLE 命令中的各语句效果如下所示。这儿展示的仅仅是报表最后一页的底部部分。

注意，系统日期是按 TO_CHAR 函数的日期格式显示的

```
                    Confidential          13-FEB-00
.
.
25 rows selected.
```

有时，写报表时会发现需要用一个更具有描述性的长标题。在这种时候，总可以使用连字符在下一行继续。然而，如果标题很长，TTITLE 或 BTITLE 命令可能就不容易读了。我们可以通过把长标题分成几段，用 SQL * Plus 的 DEFINE 命令把每小段放入用户定义的变量中来减轻这种情况。然后在 TTITLE 和 BTITLE 命令中引用这些变量。DEFINE 命令的基本语法用下面的代码进行说明：

```
DEFINE FirstLine = ' This line is one of several pieces'
DEFINE SecondLine = ' second line of a very long title'
DEFINE ThirdLine = ' As you can see we are using three user
defined variables in this example'
TITLE LEFT FirstLine CENTER SecondLine RIGHT ThirdLine SKIP
```

交互式输入报表的顶部标题和底部标题

在报表中，交互地使用替代变量、PROMPT 和 ACCEPT 命令输入顶部标题和底部标题是可能的。一个替代变量是用户变量，它由 1 个或 2 个 & 号开始。用 1 个或 2 个 & 号的替代变量的作用是不同的，下面将对其进行解释。

PROMPT 命令用来为用户显示信息。这个命令的基本语法如下所示。

```
PROMPT text-to-be-displayed-to-user
```

ACCEPT 命令有双重效用。首先，它用信息提示用户。这信息通常是指导用户键入一个值。其次，它将用户输入的值存到替代变量中。该命令的基本语法如下：

```
ACCEPT substitution-variable PROMPT text-to-user
```

下面的例子说明了这些命令的用法。

例 4

写一个报告，列出 SC 公司的所有雇员的姓名。雇员按部门分组。交互式地输入报表的标题。

我们把这些命令分成两组。第 1 组包含有 PROMPT 和 ACCEPT 命令。第 2 组包含 BAEAK 命令和创建报表的查询语句。现仅解释第 1 组命令。

```
CLEAR COLUMNS
PROMPT Please enter a title of 30 characters or less;
ACCEPT title_var PROMPT 'enter title:';
TTITLE CENTER title_var RIGHT SQL.PNO SKIP 2;

BREAK on department SKIP
SELECT D.name, E.last_name, E.first_name
FROM s_dept D, s_emp E
WHERE E.dept_id = D.id
ORDER BY D.name;
```

PROMPT 指令将显示信息给用户，告知在输入表头标题时，可以输入的最大字符数。在这例子中选择了 30 个字符，但是也可以选择任何其他合适的值

ACCEPT 命令指示用户输入标题。用户输入的标题存储在替代变量 title-var 中。然后在 TTITLE 命令中使用该变量。

为交互式地把数据输进表中，也可以用前面带有 1 个或 2 个 & 号的替代变量。当替代变量前以 1 个 & 号开始时，系统将请求用户输入一个值。当 2 个 & 号在替代变量前面时，系统要求用户输入第一次遇到的值，以后系统每次遇到这个带双 & 号的替代变量时，都使用当前值，不再要求用户输入新值　注意，使用 1 个或 2 个 & 号有很大不同　若变量以一个 & 号开始，则不管用户为该变量输入了多少个值，系统总还要用户再输入一个值。以 2

个 & 号开始的变量强迫系统仅要求用户输入一次变量值，以后系统使用该变量的当前值。

无论什么时候，当需要将字符型数据输入到替代变量中时，都有必要用单引号将其括起来。假如用户将替代变量用单引号括起来，就可以避免输入单引号的烦琐工作。替代变量的这个特性在将数据输入到表中时特别有用。下面例子，说明替代变量在输入数据时的应用。

例 5

用替代变量将数据插入到表 s_dept 中。

```
SQL> INSERT INTO s_dept VALUES (&id, &name, &region_id);

Enter value for id: 12
Enter value for name: Investigations
Enter value for region_id: 2

old   1: INSERT INTO s_dept VALUES (&id, &name, &region_id)
new   1: INSERT INTO s_dept VALUES (12, Investigations, 2)
INSERT INTO s_dept VALUES (12, Investigations, 2)
                           *
ERROR at line 1:
ORA-00984: column not allowed here
```

系统要用户为不同替代变量输入值

id 是字符型列。它需要用引号括起来。没提供时系统返回一个错误

注意，当输入数据时，系统显示所有替代变量的老值和新值。这是 Orade 的校验功能。用户发出 SET VERIFY OFF 命令（缺省为 ON），将取消这一功能。观察当输入字符型数据时，用单引号括起替代变量的效果。

```
SQL> INSERT INTO s_dept VALUES ('&id', '&name', '&region_id');

Enter value for id: 12
Enter value for name: Investigations
Enter value for region_id: 2

old   1: INSERT INTO s_dept VALUES ('&id', '&name', '&region_id')
new   1: INSERT INTO s_dept VALUES ('12', 'Investigations', '2')

1 row created.
```

观察引号括起来的替代变量

对数据值使用引号可以避免出现错误

译 后 记

Schaum系列丛书是美国计算机方面的畅销书。本书译自“Fundamentals of SQL Programming”，它是Schaum系列丛书之一。书中讲述的SQL（Structured Query Language）语言是通用的关系数据库的国际标准数据库语言。SQL成为国际标准语言以后，各个数据库厂家推出的数据库管理系统都以标准SQL作为共同的数据访问语言和标准接口，虽然不同系统对标准SQL有不同的扩充，但仍使不同的数据库系统之间的互操作有了共同的基础。对于数据库以外的领域，SQL也产生了很大的影响。有不少的软件产品将SQL语言的查询功能与图形处理、软件开发工具、人工智能系统结合起来。因此，在信息化的社会，掌握SQL语言的使用一定会受益匪浅。

本书根据最新的国际标准SQL/92（或者SQL2）版本，系统详细地介绍了SQL标准中各语句的语法、语法成分的功能和用法并指出与某些关系系统的不同点。

全书共分8章和一个手册性质的附录。第1章介绍了SQL语言发展的历史以及标准SQL语言的特点及基本概念。为了便于读者更好地理解SQL，本章有重点地、简洁地介绍了关系数据库管理系统、关系数据库和关系运算等基本概念和基本的关系理论，为后面章节的学习做好了知识准备。同时在这章中，还讲解了SQL中CREATE命令和SELECT命令的使用，并建立了样本库。第2章讨论了关系运算符在SQL语言中的实现。第3章讨论了在SELECT语句中布尔运算符的使用以及字符型数据的模糊查询问题。第4章是算术运算和SQL提供的内部函数。第5章讲解用于统计的分组函数。第6章解决了日期和时间信息的处理问题。第7章深入讨论了关系的复合查询和关系的集合运算。第8章是SQL标准提供的数据安全性问题。在附录中提供有关系数据库管理系统Oracle PC版8i的使用手册，书中所有例题的源代码都能在其上运行。还有最常用的SQL语句的语法图等信息，以便读者查阅。

书中用大量的例题讲解了SQL命令的使用及其使用应注意的细节。每章后面都有习题以及补充题。在解题过程中，指出题目的难点、重点和容易出错的地方，以加深对题目涉及内容的理解。全书共有200多个题目，包括详细步骤解答，为读者答疑解惑。

在国内图书市场上，几乎所有数据库的书籍中都会有一章讲解SQL语言。但像本书这样，根据最新国际标准SQL92，系统详尽地讲解SQL的书几乎没有，所以它是一本难得的好书。

本书可以做为高等院校数据库课程教学的理想参考书。对于从事数据库应用系统开发和进行电子商务设计的工程技术人员来说，也是一本很好的SQL/92参考手册。对于自学者更是一本难得的SQL自学教材，它可以帮助你在考试中取得好成绩。

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者

本书由李树德、胡志君和高燕林等人共同翻译完成。其中胡志君博士翻译了前四章，高燕林工程师翻译了后四章，李树德教授翻译了余下的部分并仔细审校了本书全部的译文，还改正了在原文中发现的一些小的错误。最后，请薛荣华教授审校定稿。在我们翻译的过程中，他们提出了不少宝贵的建议，对工作给予了很大的帮助，在此表示诚挚的谢意。还有我们的朋友高常荫、陈婷婷、孙奇志、阎慧娟、曹汉征、薛非和孙红志，他们帮助我们录入文稿、校对，做了很多细致的工作。另外，出版社的车立红编辑也给予了我们热情的支持。在此，一并表示我们深深的谢意。

由于时间短促，书中恐怕会有不当之处，欢迎各位专家、读者指正。

译 者